Friedrich-Alexander-Universität
Erlangen-Nürnberg

Chair for Computer Science 8
Theoretical Computer Science

Implementing Categorical Notions of Partiality and Delay in Agda

Bachelor Thesis in Computer Science

Leon Vatthauer

Advisor:

Sergey Goncharov

Friedrich-Alexander-Universität
Erlangen-Nürnberg

Erlangen, March 18, 2024

Disclaimer

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, 18. März 2024 _____

Leon Vatthauer

Abstract

Moggi famously showed how to use category theory (specifically monads) to model the semantics of effectful computations.

In this thesis we will examine how to model possibly non-terminating computations, which requires a monad supporting some form of partiality. For that we will consider categorical properties that a monad that models partiality should satisfy and then compare concrete monads in view of these properties.

Capretta's delay monad is a typical example for a partiality monad, but it comes with a too intensional notion of built-in equality. Since fixing this seems to be impossible without additional axioms, we will examine a novel approach of defining a partiality monad that works in a general setting by making use of previous research on iteration theories and drawing on the inherent connection between partiality and iteration.

Finally, we will show that in the category of setoids this partiality monad instantiates to a quotient of the delay monad, yielding a concrete description of the partiality monad in this category.

Contents

1 Introduction

Haskell is considered a purely functional programming language, though the notion of purity referenced is an informal one, not to be confused with the standard notion of pure function, which describes functions that do not have any side effects. Indeed, as a programming language that offers general recursion, Haskell does at least have to include partiality as a side effect. To illustrate this, consider the following standard list reversal function

```
reverse :: [a] -> [a]
reverse l = revAcc l []
  where
    revAcc []     a = a
    revAcc (x:xs) a = revAcc xs (x:a)
```

and regard the following definition of an infinite list

```
ones :: [Int]
ones = 1 : ones
```

Of course evaluation of the term `reverse ones` will never terminate, hence it is clear that `reverse` is a partial function. Thus, in order to reason about Haskell programs, or generally programs of any programming language offering general recursion, one needs to be able to model partiality as a side effect.

Generally for modelling programming languages there are three prevailing methods. First is the operational approach studied by Plotkin [3], where partial functions are used that map programs to their resulting values, secondly there is the denotational approach by Scott [6], where programming languages are interpreted mathematically, by functions that capture the "meaning" of programs. For this thesis we will consider the third, categorical approach that has been introduced by Moggi [5]. In the categorical approach programs are interpreted in categories, where objects represent types and monads are used to model side effects. The goal for this thesis is thus to study monads which are suitable for modeling partiality.

We use the dependently typed programming language Agda [24] as a safe and type-checked environment for reasoning in category theory, therefore in Chapter 3 we start out by quickly showcasing the Agda programming language as well as the category theory library that we will be working with. In Chapter 4 we will then consider various properties that partiality monads should satisfy and inspect Capretta's delay monad [12], which has been introduced in type theory as a coinductive data type and then studied as a monad in the category of setoids. We will examine the delay monad in a general categorical setting, where we prove strength and commutativity of this monad. However, it is not a minimal partiality monad, i.e. one that captures no other side effect besides some form of non-termination, since the monad comes with a too intensional notion of equality. In order to achieve minimality one can consider the quotient of the delay monad where a less intensional notion of equality is used. However, it is believed to be impossible to show that the monadic structure is preserved under such a quotient. In [16] the axiom of countable choice has been identified as a sufficient assumption under which the monad structure is preserved.

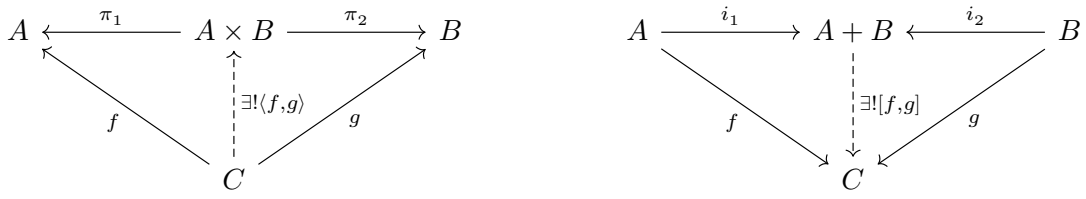
In order to define a partiality monad using no such assumptions, we will draw on the inherent connection between iteration and recursion in Chapter 5 to define a suitable partiality monad, by relating to previous research on iteration theories. This monad has first been introduced and studied in [19] under weaker assumptions than we use, concretely by weakening the notion of Elgot algebra to the notion of uniform iteration algebra, which uses fewer axioms. During mechanization of the results concerning this monad it turned out that under the weaker assumptions, desirable properties like commutativity seem not to be provable, resulting in our adaptation of this monad. Lastly, in Chapter 6 we will study this partiality monad in the category of setoids, where notably the axiom of countable choice is provable. In this category, the partiality monad turns out to be equivalent to a certain quotient of the delay monad.

2 Preliminaries

We assume familiarity with basic categorical notions, in particular: categories, functors, functor algebras and natural transformations, as well as special objects like (co)products, terminal and initial objects and special classes of morphisms like isomorphisms (isos), epimorphisms (epis) and monomorphisms (monos). In this chapter we will introduce notation that will be used throughout the thesis and also introduce some notions that are crucial to this thesis in more detail. We write $|\mathscr{C}|$ for the objects of a category $\mathscr{C}$, id_X for the identity morphism on X, $(-) \circ (-)$ for the composition of morphisms and $\mathscr{C}(X, Y)$ for the set of morphisms between X and Y. We will also sometimes omit indices of the identity and of natural transformations in favor of readability.

2.1 Distributive and Cartesian Closed Categories

Let us first introduce notation for binary (co)products by giving their usual diagrams:

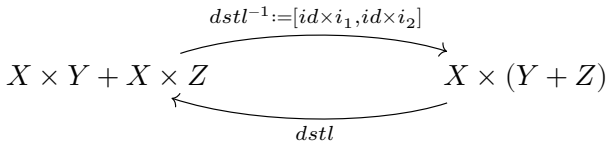


We will furthermore overload this notation and write $f \times g := \langle f \circ \pi_1, g \circ \pi_2 \rangle$ and $f + g := [i_1 \circ f, i_2 \circ g]$ on morphisms. To avoid parentheses we will use the convention that products bind stronger than coproducts.

We write 1 for the terminal object together with the unique morphism $! : A \to 1$ and 0 for the initial object with the unique morphism $¡ : A \to 0$.

Categories with finite products (i.e. binary products and a terminal object) are also called Cartesian and categories with finite coproducts (i.e. binary coproducts and an initial object) are called coCartesian.

Definition 2.1 (Distributive Category)**.** A Cartesian and coCartesian category $\mathscr{C}$ is called distributive if the canonical (left) distributivity morphism $dstl^{-1}$ is an isomorphism:



Remark 2.2. Definition 2.1 can equivalently be expressed by requiring that the canonical right

distributivity morphism is an iso, giving these inverse morphisms:

$$Y \times X + Z \times X \underset{dstr}{\overset{dstr^{-1} := [i_1 \times id, i_2 \times id]}{\rightleftarrows}} (Y + Z) \times X$$

These two can be derived from each other by taking either

$$dstr := (swap + swap) \circ dstl \circ swap$$

or

$$dstl := (swap + swap) \circ dstr \circ swap$$

where $swap := \langle \pi_2, \pi_1 \rangle : A \times B \to B \times A$.

Proposition 2.3. *The distribution morphisms can be viewed as natural transformations i.e. they satisfy the following diagrams:*

$$\begin{array}{ccc} X \times (Y + Z) & \xrightarrow{f \times (g+h)} & A \times (B + C) \\ \downarrow{\scriptstyle dstl} & & \downarrow{\scriptstyle dstl} \\ X \times Y + X \times Z & \xrightarrow{f \times g + f \times h} & A \times B + A \times C \end{array} \qquad \begin{array}{ccc} (Y + Z) \times X & \xrightarrow{(g+h) \times f} & (B + C) \times A \\ \downarrow{\scriptstyle dstr} & & \downarrow{\scriptstyle dstr} \\ Y \times X + Z \times X & \xrightarrow{g \times f + h \times f} & B \times A + C \times A \end{array}$$

Proof. We will prove naturality of $dstl$, naturality for $dstr$ is symmetric. We use the fact that $dstl^{-1}$ is an iso and therefore also an epi.

$$\begin{aligned} & dstl \circ (f \times (g + h)) \circ dstl^{-1} \\ = {} & dstl \circ (f \times (g + h)) \circ [id \times i_1, id \times i_2] \\ = {} & dstl \circ [f \times ((g + h) \circ i_1), f \times ((g + h) \circ i_2)] \\ = {} & dstl \circ [f \times (i_1 \circ g), f \times (i_2 \circ h)] \\ = {} & dstl \circ [id \times i_1, id \times i_2] \circ (f \times g + f \times h) \\ = {} & dstl \circ dstl^{-1} \circ (f \times g + f \times h) \\ = {} & (f \times g + f \times h) \\ = {} & (f \times g + f \times h) \circ dstl \circ dstl^{-1} \end{aligned}$$

□

Proposition 2.4. *The distribution morphisms satisfy the following properties:*

1. $dstl \circ (id \times i_1) = i_1$
2. $dstl \circ (id \times i_2) = i_2$
3. $[\pi_1, \pi_1] \circ dstl = \pi_1$
4. $(\pi_2 + \pi_2) \circ dstl = \pi_2$
5. $dstl \circ swap = (swap + swap) \circ dstr$
6. $dstr \circ (i_1 \times id) = i_1$
7. $dstr \circ (i_2 \times id) = i_2$
8. $(\pi_1 + \pi_1) \circ dstr = \pi_1$
9. $[\pi_2, \pi_2] \circ dstr = \pi_2$

10. $dstr \circ swap = (swap + swap) \circ dstl$

Proof. Let us verify the five properties concerning *dstl*, the ones concerning *dstr* follow symmetrically:

1.
$$\begin{aligned} & dstl \circ (id \times i_1) \\ = {} & dstl \circ [id \times i_1, id \times i_2] \circ i_1 \\ = {} & dstl \circ dstl^{-1} \circ i_1 \\ = {} & i_1 \end{aligned}$$

2.
$$\begin{aligned} & dstl \circ (id \times i_2) \\ = {} & dstl \circ [id \times i_1, id \times i_2] \circ i_2 \\ = {} & dstl \circ dstl^{-1} \circ i_2 \\ = {} & i_2 \end{aligned}$$

3.
$$\begin{aligned} & \pi_1 \\ = {} & \pi_1 \circ dstl^{-1} \circ dstl \\ = {} & [\pi_1 \circ (id \times i_1), \pi_1 \circ (id \times i_2)] \circ dstl \\ = {} & [\pi_1, \pi_1] \circ dstl \end{aligned}$$

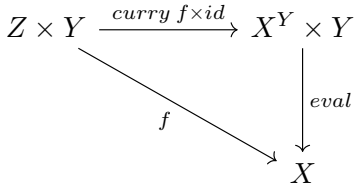
4.
$$\begin{aligned} & \pi_2 \\ = {} & \pi_2 \circ dstl^{-1} \circ dstl \\ = {} & [\pi_2 \circ (id \times i_1), \pi_2 \circ (id \times i_2)] \circ dstl \\ = {} & (\pi_2 + \pi_2) \circ dstl \end{aligned}$$

5.
$$\begin{aligned} & dstl \circ swap \\ = {} & dstl \circ swap \circ dstr^{-1} \circ dstr \\ = {} & dstl \circ [swap \circ (i_1 \times id), swap \circ (i_2 \times id)] \circ dstr \\ = {} & dstl \circ [(id \times i_1) \circ swap, (id \times i_2) \circ swap] \circ dstr \\ = {} & dstl \circ [id \times i_1, id \times i_2] \circ (swap + swap) \circ dstr \\ = {} & dstl \circ dstl^{-1} \circ (swap + swap) \circ dstr \\ = {} & (swap + swap) \circ dstr \end{aligned}$$

□

Definition 2.5 (Exponential Object)**.** Let $\mathscr{C}$ be a Cartesian category and $X, Y \in |\mathscr{C}|$. An object X^Y is called an exponential object (of X and Y) if there exists an evaluation morphism $eval : X^Y \times Y \to X$ and for any $f : X \times Y \to Z$ there exists a morphism $curry\ f : X \to Z^Y$

that is unique with respect to the following diagram:



Proposition 2.6. *Every exponential object X^Y satisfies the following properties:*

1. *The mapping $curry : \mathscr{C}(X \times Y, Z) \to \mathscr{C}(X \to Z^Y)$ is injective,*
2. *$curry(eval \circ (f \times id)) = f$ for any $f : X \times Y \to Z$,*
3. *$curry\ f \circ g = curry(f \circ (g \times id))$ for any $f : X \times Y \to Z, g : A \to X$.*

Proof. 1. Let $f, g : X \times Y \to Z$ and $curry\ f = curry\ g$, then indeed

$$f = eval \circ (curry\ f \times id) = eval \circ (curry\ g \times id) = g.$$

2. $curry(eval \circ (f \times id)) = f$ follows instantly by uniqueness of $curry(eval \circ (f \times id))$.
3. Note that $eval \circ (curry\ f \circ g \times id) = eval \circ (curry\ f \times id) \circ (g \times id) = f \circ (g \times id)$, thus we are done by uniqueness of $curry(f \circ (g \times id))$. □

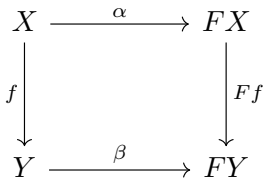
A Cartesian closed category is a Cartesian category $\mathscr{C}$ that also has an exponential object X^Y for any $X, Y \in |\mathscr{C}|$. The internal logic of Cartesian closed categories is the simply typed λ-calculus, which makes them a suitable environment for interpreting programming languages. For the rest of this thesis we will work in an ambient distributive category $\mathscr{C}$, that however need not be Cartesian closed as to be more general.

2.2 F-Coalgebras

Let $F : \mathscr{C} \to \mathscr{C}$ be an endofunctor. Recall that F-algebras are tuples $(X, \alpha : FX \to X)$ consisting of an object of $\mathscr{C}$ and a morphism out of the functor. Initial F-algebras have been studied extensively as a means of modeling inductive data types together with induction and recursion principles [7]. For this thesis we will be more interested in the dual concept namely terminal coalgebras; let us formally introduce them now.

Definition 2.7 (F-Coalgebra)**.** A tuple $(X \in |\mathscr{C}|, \alpha : X \to FX)$ is called an *F-coalgebra* (hereafter referred to as just *coalgebra*).

Definition 2.8 (Coalgebra Morphisms)**.** Let $(X, \alpha : X \to FX)$ and $(Y, \beta : Y \to FY)$ be two coalgebras. A morphism between these coalgebras is a morphism $f : X \to Y$ such that the following diagram commutes:



Coalgebras on a given functor together with their morphisms form a category that we call $Coalgs(F)$.

Proposition 2.9. *Coalgs(F) is a category.*

Proof. Let $(X, \alpha : X \to FX)$ be a coalgebra. The identity morphism on (X, α) is the identity morphism of $\mathscr{C}$ that trivially satisfies $\alpha \circ id = Fid \circ \alpha$.

Let $(X, \alpha : X \to FX), (Y, \beta : Y \to FY)$ and $(Z, \gamma : Z \to FZ)$ be coalgebras. Composition of $f : (X, \alpha) \to (Y, \beta)$ and $g : (Y, \beta) \to (Z, \gamma)$ is composition of the underlying morphisms in $\mathscr{C}$ where:

$$\begin{aligned} & \gamma \circ g \circ f \\ = & Fg \circ \beta \circ f \\ = & Fg \circ Ff \circ \alpha \\ = & F(g \circ f) \circ \alpha \end{aligned}$$

□

The terminal object of *Coalgs(F)* is sometimes called *final coalgebra*, we will however call it the *terminal coalgebra* for consistency with initial F-algebras. Similarly to initial F-algebras, the final coalgebra can be used for modeling the semantics of coinductive data types where terminality of the coalgebra yields corecursion as a definitional principle and coinduction as a proof principle. Let us make the universal property of terminal coalgebras concrete.

Definition 2.10 (Terminal Coalgebra)**.** A coalgebra $(T, t : T \to FT)$ is called a terminal coalgebra if for any other coalgebra $(X, \alpha : X \to FX)$ there exists a unique morphism $[\![\alpha]\!] : X \to T$ satisfying:

$$\begin{array}{ccc} X & \xrightarrow{\alpha} & FX \\ {\scriptstyle [\![\alpha]\!]}\downarrow & & \downarrow{\scriptstyle F[\![\alpha]\!]} \\ T & \xrightarrow{t} & FT \end{array}$$

We use the common notation νF to denote the terminal coalgebra for F (if it exists).

We will discuss the concrete form that induction and coinduction take in a type theory in Chapter 3. Let us now reiterate a famous Lemma concerning terminal F-coalgebras.

Lemma 2.11 (Lambek's Lemma [1])**.** *Let $(T, t : T \to FT)$ be a terminal coalgebra. Then t is an isomorphism.*

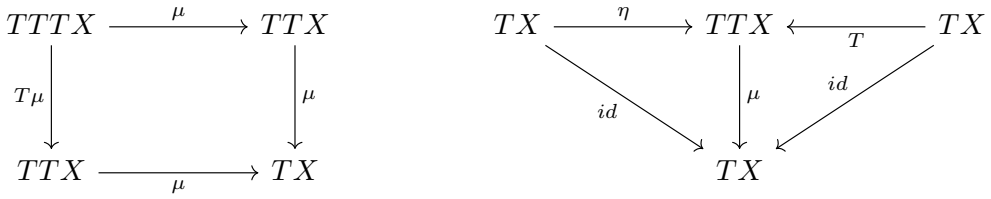
2.3 Monads

Monads are widely known in functional programming as a means for modeling effects in "pure" languages and are also central to this thesis. Let us recall the basic definitions[2][5].

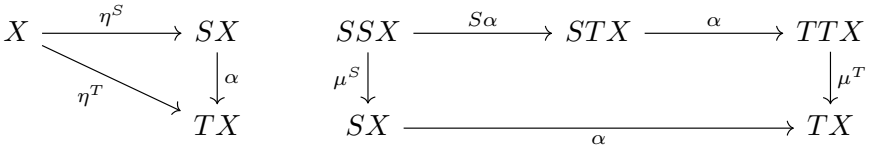
Definition 2.12 (Monad)**.** A monad **T** on a category $\mathscr{C}$ is a triple (T, η, μ), where $T : \mathscr{C} \to \mathscr{C}$ is an endofunctor and $\eta : Id \to T, \mu : TT \to T$ are natural transformations, satisfying the following laws:

$$\mu_X \circ \mu_{TX} = \mu_X \circ T\mu_X \tag{M1}$$
$$\mu_X \circ \eta_{TX} = id_{TX} \tag{M2}$$
$$\mu_X \circ T\eta_X = id_{TX} \tag{M3}$$

These laws are expressed by the following diagrams:



Definition 2.13 (Monad Morphism)**.** A morphism between monads $(S : \mathscr{C} \to \mathscr{C}, \eta^S, \mu^S)$ and $(T : \mathscr{C} \to \mathscr{C}, \eta^T, \mu^T)$ is a natural transformation $\alpha : S \to T$ between the underlying functors such that the following diagrams commute.



This yields a category of monads on a given category $\mathscr{C}$ that we call *Monads*($\mathscr{C}$).

Proposition 2.14. *Monads*($\mathscr{C}$) *is a category.*

Proof. The identity morphism of *Monads*($\mathscr{C}$) is the identity natural transformation $Id : F \to F$, which trivially respects the monad unit and multiplication. Composition of monad morphisms is composition of the underlying natural transformation, the diagrams then also follow easily. □

Monads can also be specified in a second equivalent way that is better suited to describe computation.

Definition 2.15 (Kleisli Triple)**.** A Kleisli triple on a category $\mathscr{C}$ is a triple $(F, \eta, (-)^*)$, where $F : |C| \to |C|$ is a mapping on objects, $(\eta_X : X \to FX)_{X \in |C|}$ is a family of morphisms and for every morphism $f : X \to FY$ there exists a morphism $f^* : FX \to FY$ called the Kleisli lifting, where the following laws hold:

$$\eta^*_X = id_{FX} \tag{K1}$$
$$f^* \circ \eta_X = f \quad \text{for any } f : X \to FY \tag{K2}$$
$$f^* \circ g* = (f^* \circ g)^* \quad \text{for any } f : Y \to FZ, g : X \to FY \tag{K3}$$

Let $f : X \to TY, g : Y \to TZ$ be two programs, where T is a Kleisli triple. These programs can be composed by taking: $f^* \circ g : X \to TZ$, which is called Kleisli composition. Haskell's do-notation is a useful tool for writing Kleisli composition in a legible way. We will sometimes express $(f^* \circ g)x$ equivalently as

```
do y <- g x
   f y
```

This yields the category of programs for a Kleisli triple that is called the Kleisli category.

Definition 2.16 (Kleisli Category)**.** Given a monad T on a category $\mathscr{C}$, the Kleisli category $\mathscr{C}^T$ is defined as:

- $|\mathscr{C}^T| = |C|$

- $\mathscr{C}^T(X,Y) = \mathscr{C}(X,TY)$
- Composition of programs is Kleisli composition.
- The identity morphisms are the unit morphisms of T, $id_X = \eta_X : X \to TX$

The laws of categories then follow from the Kleisli triple laws.

Proposition 2.17 ([4])**.** *The notions of Kleisli triple and monad are equivalent.*

Proof. The crux of this proof is defining the triples, the proofs of the corresponding laws (functoriality, naturality, monad and Kleisli triple laws) are left out.

"$\Rightarrow$": Given a Kleisli triple $(F, \eta, (-)^*)$, we obtain a monad (F, η, μ) where F is the object mapping of the Kleisli triple together with the functor action $F(f : X \to Y) = (\eta_Y \circ f)^*$, η is the morphism family of the Kleisli triple where naturality is easy to show and μ is a natural transformation defined as $\mu_X = id^*_{FX}$

"$\Leftarrow$":
Given a monad (F, η, μ), we obtain a Kleisli triple $(F, \eta, (-)^*)$ by restricting the functor F on objects, taking the underlying mapping of η and defining $f^* = \mu_Y \circ Ff$ for any $f : X \to FY$. □

For the rest of this thesis we will use both equivalent notions interchangeably to make definitions easier.

2.4 Strong and Commutative Monads

Consider the following program in do-notation

```
do y <- g x
   f (x , y)
```

where $g : X \to TY$ and $f : X \times Y \to TZ$ are programs and $\mathbf{T}$ is a monad. Kleisli composition does not suffice for interpreting this program, we will get stuck at

$$X \xrightarrow{\langle id, g\rangle} X \times TY \xrightarrow{?} T(X \times Y) \xrightarrow{f^*} TZ.$$

Instead, one needs the following stronger notion of monad.

Definition 2.18 (Strong Monad)**.** A monad (T, η, μ) on a Cartesian category $\mathscr{C}$ is called strong if there exists a natural transformation $\tau_{X,Y} : X \times TY \to T(X \times Y)$ that satisfies the following conditions:

$$T\pi_2 \circ \tau_{1,X} = \pi_2 \tag{S1}$$
$$\tau_{X,Y} \circ (id_X \times \eta_Y) = \eta_{X\times Y} \tag{S2}$$
$$\tau_{X,Y} \circ (id_X \times \mu_Y) = \mu_{X\times Y} \circ T\tau_{X,Y} \circ \tau_{X,TY} \tag{S3}$$
$$M\alpha_{X,Y,Z} \circ \tau_{X\times Y,Z} = \tau_{X,Y\times Z} \circ (id_X \times \tau_{Y,Z}) \circ \alpha_{X,Y,TZ} \tag{S4}$$

where $\alpha_{X,Y,Z} = \langle\langle \pi_1, \pi_1 \circ \pi_2\rangle, \pi_2 \circ \pi_2\rangle : X \times (Y \times Z) \to (X \times Y) \times Z$ is the associativity morphism on products.

Definition 2.19 (Strong Monad Morphism)**.** A morphism between two strong monads $(S : \mathscr{C} \to \mathscr{C}, \eta^S, \mu^S, \tau^S)$ and $(T : \mathscr{C} \to \mathscr{C}, \eta^T, \mu^T, \tau^T)$ is a morphism between monads as in Definition 2.13 where additionally the following diagram commutes.

$$\begin{array}{ccc} X \times SY & \xrightarrow{id \times \alpha} & X \times TY \\ \downarrow{\scriptstyle \tau^S} & & \downarrow{\scriptstyle \tau^T} \\ S(X \times Y) & \xrightarrow{\alpha} & T(X \times Y) \end{array}$$

As with monads this yields a category of strong monads on $\mathscr{C}$ that we call $\mathit{StrongMonads}(\mathscr{C})$.

Let us now consider the following two programs

```
do x <- p
   y <- q
   return (x, y)
```

```
do y <- q
   x <- p
   return (x, y)
```

Where $p : TX$ and $q : TY$ are computations of some monad T. A monad where these programs are equal, is called commutative.

Definition 2.20 (Commutative Monad)**.** A strong monad $\mathbf{T}$ is called commutative if the (right) strength τ commutes with the induced left strength

$$\sigma_{X,Y} = T\mathit{swap} \circ \tau_{Y,X} \circ \mathit{swap} : TX \times Y \to T(X \times Y)$$

that satisfies symmetrical conditions to the ones τ satisfies. Concretely, $\mathbf{T}$ is called commutative if the following diagram commutes:

$$\begin{array}{ccc} TX \times TY & \xrightarrow{\tau} & T(TX \times Y) \\ \downarrow{\scriptstyle \sigma} & & \downarrow{\scriptstyle \sigma^*} \\ T(X \times TY) & \xrightarrow[\tau^*]{} & T(X \times Y) \end{array}$$

2.5 Free Objects

Free objects, roughly speaking, are constructions for instantiating structure declarations in a minimal way. We will rely on free structures in Chapter 5 to define a monad in a general setting. We recall the definition to establish some notation and then describe how to obtain a monad via existence of free objects.

Definition 2.21 (Free Object)**.** Let $\mathscr{C}, \mathscr{D}$ be categories and $U : \mathscr{C} \to \mathscr{D}$ be a forgetful functor (whose construction usually is obvious). A free object on some object $X \in |\mathscr{D}|$ is an object $FX \in |\mathscr{C}|$ together with a morphism $\eta : X \to UFX$ such that for any $Y \in |\mathscr{C}|$ and $f : X \to UY$ there exists a unique morphism $f^\star : FX \to Y$ satisfying:

$$\begin{array}{ccc} X & \xrightarrow{f} & UY \\ {\scriptstyle \eta}\downarrow & \nearrow_{Uf^\star} & \\ UFX & & \end{array}$$

Proposition 2.22. *Let $U : \mathscr{C} \to \mathscr{D}$ be a forgetful functor. If for every $X \in |\mathscr{D}|$ a free object $FX \in |C|$ exists then $(X \mapsto UFX, \eta : X \to UFX, (f : X \to UFY)^\star : UFX \to UFY)$ is a Kleisli triple on $\mathscr{D}$.*

Proof. We are left to check the laws of Kleisli triples.

(K1) $\eta^\star = id$

By uniqueness of $\eta^\star$ it suffices to show that $id \circ \eta = \eta$ which holds trivially.

(K2) $f^\star \circ \eta = f$ for any $f : X \to UFY$

This is the universal property concerning $f^\star$.

(K3) $f^\star \circ g^\star = (f^\star \circ g)^\star$ for any $f : Y \to UFZ, g : X \to UFY$

By uniqueness of $(f^\star \circ g)^\star$ we are left to show $f^\star \circ g^\star \circ \eta = f^\star \circ g$ which again follows directly by the universal property of $g^\star$. □

3 Implementing Category Theory in Agda

There are many formalizations of category theory in proof assistants like Coq or Agda. The benefits of such a formalization are clear: having a usable formalization allows one to reason about categorical notions in a type checked environment that makes errors less likely. Ideally such a development will bring researchers together and enable them to work in a unified setting that enables efficient communication of ideas and concepts. In this thesis we will work with the dependently typed programming language Agda [24] and the agda-categories [20] library that serves as an extensive foundation of categorical definitions. This chapter shall serve as a quick introduction to the relevant parts of Agda's type theory, the agda-categories library, and the formalization of this thesis.

3.1 The Underlying Type Theory

Agda implements a Martin-Löf style dependent type theory with *inductive* and *coinductive types* as well as an infinite hierarchy of universes `Set₀, Set₁, …`, where usually `Set₀` is abbreviated as `Set`. Recall that inductive types usually come with a principle for defining functions from inductive types, called *recursion* and a principle for proving facts about the inhabitants of inductive types, called *induction*. These are standard notions and need no further introduction. Coinductive types come with dual principles that are however lesser known. Dually to inductive types that are defined by their *constructors*, coinductive types are defined by their *destructors* or their observational behavior. Take the type of streams over a type `A`, for example. In Agda one would define this type as a coinductive record like so:

```
record Stream (A : Set) : Set where
  coinductive
  field
    head : A
    tail : Stream A
```

i.e. the type of streams over `A` is defined by the two destructors `head : Stream A → A` and `tail : Stream A → Stream A` that return the head and the tail of the stream respectively. Now, *corecursion* is a principle for defining functions into coinductive types by specifying how results of the function may be observed. Take for example the following function which defines an infinite stream repeating the same argument and is defined by use of Agda's *copatterns*.

```
repeat : {A : Set} (a : A) → Stream A
head (repeat a) = a
tail (repeat a) = repeat a
```

Let us introduce the usual notion of stream bisimilarity. Given two streams, they are bisimilar if their heads are equal and their tails are bisimilar.

```
record _≈_ {A} (s : Stream A) (t : Stream A) : Set where
  coinductive
```

```
  field
    head : head s ≡ head t
    tail : tail s ≈ tail t
```

In this definition `_≡_` is the built-in propositional equality in Agda with the single constructor `refl`. We can now use coinduction as a proof principle to proof a fact about streams.

```
repeat-eq : ∀ {A} (a : A) → repeat a ≈ tail (repeat a)
head (repeat-eq {A} a) = refl
tail (repeat-eq {A} a) = repeat-eq a
```

Where in the coinductive step we were able to assume that `repeat a ≈ tail(repeat a)` already holds and showed that thus `tail(repeat a) ≈ tail(tail(repeat a))` holds.

Streams are always infinite and thus this representation of coinductive types as coinductive records is well suited for them. However, consider the type of *possibly* infinite lists, that we will call `coList`. In pseudo notation this type can be defined as

```
codata coList (A : Set) : Set where
  nil : coList A
  _::_ : A → coList A → coList A
```

That is, the coinductive type `coList` is defined by the constructors `nil` and `_::_`. Agda does implement a second way of defining coinductive types that allows exactly such definitions, however the use of these sometimes called *positive coinductive types* is discouraged, since it is known to break subject reduction [22][23]. Instead, sticking to coinductive records, we can define `coList` as two mutual types, one inductive and the other coinductive:

```
mutual
  data coList (A : Set) : Set where
    nil : coList A
    _::_ : A → coList′ A → coList A
  record coList′ (A : Set) : Set where
    coinductive
    field force : coList A
```

Unfortunately, this does add the overhead of having to define functions on `coList` as mutual recursive functions, e.g. the `repeat` function from before can be defined as

```
mutual
  repeat  : {A : Set} (a : A) → coList A
  repeat′ : {A : Set} (a : A) → coList′ A
  repeat a = a :: repeat′ a
  force (repeat′ a) = repeat a
```

or more succinctly using a λ-function

```
repeat : {A : Set} (a : A) → coList A
repeat a = a :: λ { .force → repeat a }
```

In Chapter 6 we will work with such a coinductive type that is defined by constructors, hence to avoid the overhead of defining every data type twice and using mutual function definitions in the thesis, we will work in a type theory that does offer coinductive types with constructors and their respective corecursion and coinduction principles. However, in the formalization we stick to using coinductive records as to implement best practices. The translation between the two styles is straightforward, as illustrated by the previous example.

3.2 Setoid Enriched Categories

Let us now consider how to implement category theory in Agda. The usual textbook definition of a category glosses over some design decisions that have to be made when implementing it in type theory. One would usually see something like this:

Definition 3.1 (Category)**.** A category consists of

- A collection of objects
- A collection of morphisms between objects
- For every two morphisms $f : X \to Y, g : Y \to Z$ another morphism $g \circ f : X \to Z$ called the composition
- For every object X a morphism $id_X : X \to X$ called the identity

where the composition is associative, and the identity morphisms are identities with respect to the composition.

Here *collection* refers to something that behaves set-like, which is not a set and is needed to prevent size issues (there is no set of all sets, otherwise we would obtain Russel's paradox, but there is a collection of all sets), it is not immediately clear how to translate this to type theory. Furthermore, in mathematical textbooks equality between morphisms is usually taken for granted, i.e. there is some global notion of equality that is clear to everyone. In type theory we need to be more thorough as there is no global notion of equality, eligible for all purposes, e.g. the standard notion of propositional equality has issues when dealing with functions in that it requires extra axioms like functional extensionality.

The definition of category that we will work with can be seen in Listing 1 (unnecessary information has been stripped). The key differences to the definition above are firstly that instead of talking about collections, Agda's infinite `Set` hierarchy is utilized to prevent size issues. This notion of category is thus parametrized by 3 universe levels, one for objects, one for morphisms and one for equalities. A consequence is that the category does not contain a type of all morphisms, instead it contains a type of morphisms for any pair of objects. Furthermore, the types of morphisms are equipped with an equivalence relation `_≈_`, making them setoids. This addresses the aforementioned issue of how to implement equality between morphisms: the notion of equality is just added to the definition of a category. This version of the notion of category is also called a *setoid-enriched category*.

As a consequence of using a custom equality relation, proofs like `∘-resp-≈` are needed throughout the library to make sure that operations on morphisms respect the equivalence relation. In the thesis we will omit such proofs, but they are contained in our formalization. Lastly, the designers of agda-categories also include symmetric proofs like `sym-assoc` to definitions, in this case to guarantee that the opposite category of the opposite category is equal to the original category, and a similar reason for requiring `identity²`, we won't address the need for these proofs and just accept the requirement as given for the rest of the thesis.

```
record Category (o ℓ e : Level) : Set (suc (o ⊔ ℓ ⊔ e)) where
  field
    Obj : Set o
    _⇒_ : Obj → Obj → Set ℓ
    _≈_ : ∀ {A B : Obj } → (A ⇒ B) → (A ⇒ B) → Set e

    id  : ∀ {A : Obj} → (A ⇒ A)
    _∘_ : ∀ {A B C : Obj} → (B ⇒ C) → (A ⇒ B) → (A ⇒ C)

    assoc     : ∀ {A B C D} {f : A ⇒ B} {g : B ⇒ C} {h : C ⇒ D}
      → (h ∘ g) ∘ f ≈ h ∘ (g ∘ f)
    sym-assoc : ∀ {A B C D} {f : A ⇒ B} {g : B ⇒ C} {h : C ⇒ D}
      → h ∘ (g ∘ f) ≈ (h ∘ g) ∘ f
    identity¹ : ∀ {A B} {f : A ⇒ B} → id ∘ f ≈ f
    identityʳ : ∀ {A B} {f : A ⇒ B} → f ∘ id ≈ f
    identity² : ∀ {A} → id ∘ id {A} ≈ id {A}
    equiv     : ∀ {A B} → IsEquivalence (_≈_ {A} {B})
    ∘-resp-≈  : ∀ {A B C} {f h : B ⇒ C} {g i : A ⇒ B}
      → f ≈ h
      → g ≈ i
      → f ∘ g ≈ h ∘ i
```

Listing 1: Definition of Category [20]

From this it should be clear how other basic notions like functors or natural transformations look in the library.

3.3 The formalization

Every result and used fact (except for Proposition 4.4) in this thesis has been proven either in our own formalization[1] or in the agda-categories library [20]. The formalization is meant to be used as a reference alongside this thesis, where concrete details of proofs can be looked up and verified. The preferred format for viewing the formalization is as automatically generated clickable HTML code[2], where multiple annotations explaining the structure have been added in Markdown, however concrete explanations of the proofs and their main ideas are mostly just contained in this thesis.

In the future this formalization may be adapted into a separate library that uses the agda-categories library as a basis but is more focussed on the study of partiality monads and iteration theories. As such the formalization has been structured similar to the agda-categories library, where key concepts such as monads correspond to separate top-level folders, which contain the core definitions as well as folders for sub-concepts and their properties.

[1] https://git8.cs.fau.de/theses/bsc-leon-vatthauer
[2] https://wwwcip.cs.fau.de/ hy84coky/bsc-thesis/

4 Partiality Monads

Moggi's categorical semantics [5] describe a way to interpret an effectful programming language in a category. For this one needs a (strong) monad T capturing the desired effects, then we can take the elements of TA as denotations for programs of type A. The Kleisli category of T can be viewed as the category of programs, which gives us a way of composing programs (Kleisli composition).

For this thesis we will restrict ourselves to monads for modeling partiality, the goal of this chapter is to capture what it means to be a partiality monad and look at two common examples.

4.1 Properties of Partiality Monads

We will now look at how to express the following non-controversial properties of a minimal partiality monad categorically:

1. Irrelevance of execution order
2. Partiality of programs
3. No other effect besides some form of non-termination

The first property of course holds for any commutative monad, the other two are more interesting.

To ensure that programs are partial, we recall the following notion by Cockett and Lack [9], that axiomatizes the notion of partiality in a category:

Definition 4.1 (Restriction Structure)**.** A restriction structure on a category $\mathscr{C}$ is a mapping $dom : \mathscr{C}(X,Y) \to \mathscr{C}(X,X)$ with the following properties:

$$\begin{aligned} f \circ (\text{dom } f) &= f \\ (\text{dom } f) \circ (\text{dom } g) &= (\text{dom } g) \circ (\text{dom } f) \\ \text{dom } (g \circ (\text{dom } f)) &= (\text{dom } g) \circ (\text{dom } f) \\ (\text{dom } h) \circ f &= f \circ \text{dom } (h \circ f) \end{aligned}$$

for any $X, Y, Z \in |\mathscr{C}|, f : X \to Y, g : X \to Z, h : Y \to Z$.

The morphism dom $f : X \to X$ represents the domain of definiteness of $f : X \to Y$. In the category of partial functions this takes the following form:

$$(\text{dom } f)(x) = \begin{cases} x & \text{if } f(x) \text{ is defined} \\ \text{undefined} & \text{else} \end{cases}$$

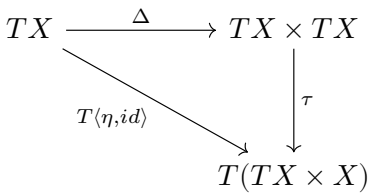
That is, dom f is only defined on values where f is defined and for those values it behaves like the identity function.

Definition 4.2 (Restriction Category)**.** Every category has a trivial restriction structure by taking $dom(f : X \to Y) = id_X$. We call categories with a non-trivial restriction structure *restriction categories*.

For a suitable defined partiality monad T the Kleisli category $\mathscr{C}^T$ should be a restriction category.

Lastly, we also recall the following notion by Bucalo et al. [11] which captures what it means for a monad to have no other side effect besides some sort of non-termination:

Definition 4.3 (Equational Lifting Monad)**.** A commutative monad T is called an *equational lifting monad* if the following diagram commutes:



where $\Delta_X : X \to X \times X$ is the diagonal morphism.

To make the equational lifting property more comprehensible we can alternatively state it using do-notation. The equational lifting property states that the following programs must be equal:

```
do x <- p
   return (x , p)
```

```
do x <- p
   return (x , return x)
```

That is, if some computation $p : TX$ terminates with the result $x : X$, then $p = return\ x$ must hold afterwards. This of course implies that running p multiple times yields the same result as running p once.

Proposition 4.4 ([11])**.** *If T is an equational lifting monad the Kleisli category $\mathscr{C}^T$ is a restriction category.*

Definition 4.3 combines all three properties stated at the beginning of the section, so when studying partiality monads in this thesis, we ideally expect them to be equational lifting monads. For the rest of this chapter we will use these definitions to compare two monads that are commonly used to model partiality.

4.2 The Maybe Monad

The endofunctor $MX = X + 1$ extends to a monad by taking $\eta_X = i_1 : X \to X + 1$ and $\mu_X = [id, i_2] : (X + 1) + 1 \to X + 1$. The monad laws follow easily. This is generally known as the maybe monad and can be viewed as the canonical example of an equational lifting monad.

Theorem 4.5. *M is an equational lifting monad.*

Proof. We define strength as

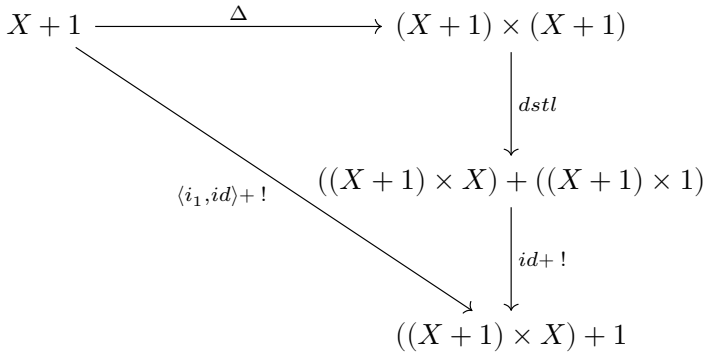
$$\tau_{X,Y} := X \times (Y + 1) \xrightarrow{dstl} (X \times Y) + (X \times 1) \xrightarrow{id+!} (X \times Y) + 1.$$

Naturality of τ follows by naturality of $dstl$

$$\begin{aligned}
&(id+!) \circ dstl \circ (id \times (f + id)) \\
&= (id+!) \circ ((id \times f) + (id \times id)) \circ dstl \\
&= ((id \times f)+!) \circ dstl \\
&= ((id \times f) + id) \circ (id+!) \circ dstl.
\end{aligned}$$

The other strength laws and commutativity can be proven by using simple properties of distributive categories, we added these proofs to the formalization for completeness.

We are thus left to check the equational lifting law:



This is easily proven by pre-composing with i_1 and i_2, indeed

$$\begin{aligned}
&(id + !) \circ dstl \circ \langle i_1, i_1 \rangle \\
&= (id + !) \circ dstl \circ (id \times i_1) \circ \langle i_1, id \rangle \\
&= (id + !) \circ i_1 \circ \langle i_1, id \rangle \\
&= i_1 \circ \langle i_1, id \rangle,
\end{aligned}$$

and

$$\begin{aligned}
&(id + !) \circ dstl \circ \langle i_2, i_2 \rangle \\
&= (id + !) \circ dstl \circ (id \times i_2) \circ \langle i_2, id \rangle \\
&= (id + !) \circ i_2 \circ \langle i_2, id \rangle \\
&= i_2 \circ ! \circ \langle i_2, id \rangle \\
&= i_2 \circ !.
\end{aligned}$$

□

In the setting of classical mathematics this monad is therefore sufficient for modeling partiality, but in general it will not be useful for modeling non-termination as a side effect, since one would need to know beforehand whether a program terminates or not. For the purpose of modeling possibly non-terminating computations another monad has been introduced by Capretta [12].

4.3 The Delay Monad

Capretta's delay monad [12] is a coinductive datatype whose inhabitants can be viewed as suspended computations. This datatype is usually defined by the two coinductive constructors $now : X \to DX$ and $later : DX \to DX$, where now lifts a value inside a computation and

later intuitively delays a computation by one time unit. See Chapter 6 for a type theoretical study of this monad. Categorically we obtain the delay monad by the terminal coalgebras $DX = \nu A.X + A$, which we assume to exist. In this section we will show that these terminal coalgebras indeed yield a monad that is strong and commutative.

Since DX is defined as a terminal coalgebra, we can define morphisms via corecursion and prove theorems by coinduction. By Lemma 2.11 the coalgebra structure $out : DX \to X + DX$ is an isomorphism, whose inverse can be decomposed into the two constructors mentioned before: $out^{-1} = [now, later] : X + DX \to DX$.

Lemma 4.6. *The following conditions hold:*

- $now : X \to DX$ *and* $later : DX \to DX$ *satisfy:*

$$out \circ now = i_1 \qquad\qquad out \circ later = i_2 \tag{D1}$$

- *For any* $f : X \to DY$ *there exists a unique morphism* $f^* : DX \to DY$ *such that the following commutes.*

$$\begin{array}{ccc} DX & \xrightarrow{\quad out \quad} & X + DX \\ {\scriptstyle f^*}\downarrow & & \downarrow{\scriptstyle [out \circ f, i_2 \circ f^*]} \\ DY & \xrightarrow{\quad out \quad} & Y + DY \end{array} \tag{D2}$$

- *There exists a unique morphism* $\tau : X \times DY \to D(X \times Y)$ *such that:*

$$\begin{array}{ccccc} X \times DY & \xrightarrow{id \times out} & X \times (Y + DY) & \xrightarrow{dstl} & X \times Y + X \times DY \\ {\scriptstyle \tau}\downarrow & & & & \downarrow{\scriptstyle id + \tau} \\ D(X \times Y) & & \xrightarrow{\qquad\qquad out \qquad\qquad} & & X \times Y + D(X \times Y) \end{array} \tag{D3}$$

Proof. We will make use of the fact that every DX is a terminal coalgebra:

(D1) These follow by definition of *now* and *later*:

$$\begin{aligned} out \circ now &= out \circ out^{-1} \circ i_1 = i_1 \\ out \circ later &= out \circ out^{-1} \circ i_2 = i_2 \end{aligned}$$

(D2) We define $f^* = [\![\alpha]\!] \circ i_1$, where $[\![\alpha]\!]$ is the unique coalgebra morphism in this diagram:

$$\begin{array}{ccc} DX & & \\ \downarrow{\scriptstyle i_1} & & \\ DX + DY & \xrightarrow{\alpha := [[[i_1, i_2 \circ i_2] \circ (out \circ f), i_2 \circ i_1] \circ out, (id + i_2) \circ out]} & Y + (DX + DY) \\ {\scriptstyle [\![\alpha]\!]}\downarrow & & \downarrow{\scriptstyle id + [\![\alpha]\!]} \\ DY & \xrightarrow{\qquad\qquad out \qquad\qquad} & Y + DY \end{array}$$

Note that $[\![\alpha]\!] \circ i_2 = id : (DY, out) \to (DY, out)$, by uniqueness of the identity morphism and the fact that $[\![\alpha]\!] \circ i_2$ is a coalgebra morphism, since

$$\begin{aligned} & out \circ [\![\alpha]\!] \circ i_2 \\ = {} & (id + [\![\alpha]\!]) \circ \alpha \circ i_2 \\ = {} & (id + [\![\alpha]\!]) \circ (id + i_2) \circ out \\ = {} & (id + [\![\alpha]\!] \circ i_2) \circ out \end{aligned}$$

Let us verify that f^* indeed satisfies the requisite property:

$$\begin{aligned}
&out \circ [\![\alpha]\!] \circ i_1 \\
&= (id + [\![\alpha]\!]) \circ \alpha \circ i_1 \\
&= (id + [\![\alpha]\!]) \circ [[i_1, i_2 \circ i_2] \circ out \circ f, i_2 \circ i_1] \circ out \\
&= [[(id + [\![\alpha]\!]) \circ i_1, (id + [\![\alpha]\!]) \circ i_2 \circ i_2] \circ out \circ f, (id + [\![\alpha]\!]) \circ i_2 \circ i_1] \circ out \\
&= [[i_1, i_2 \circ [\![\alpha]\!] \circ i_2] \circ out \circ f, i_2 \circ [\![\alpha]\!] \circ i_1] \circ out \\
&= [out \circ f, i_2 \circ f^*] \circ out.
\end{aligned}$$

We are left to check uniqueness of f^*. Let $g : DX \to DY$ and $out \circ g = [out \circ f, i_2 \circ g] \circ out$. Note that $[g, id] : DX + DY \to DY$ is a coalgebra morphism, since

$$\begin{aligned}
&out \circ [g, id] \\
&= [out \circ g, out] \\
&= [[out \circ f, i_2 \circ g] \circ out, out] \\
&= [[[i_1, i_2] \circ out \circ f, i_2 \circ g] \circ out, (id + id) \circ out] \\
&= [[[i_1, i_2 \circ [g, id] \circ i_2] \circ out \circ f, i_2 \circ [g, id] \circ i_1] \circ out, (id + [g, id] \circ i_2) \circ out] \\
&= [[[(id + [g, id]) \circ i_1, (id + [g, id]) \circ i_2 \circ i_2] \circ out \circ f, (id + [g, id]) \circ i_2 \circ i_1] \circ out \\
&\quad , (id + [g, id]) \circ (id + i_2) \circ out] \\
&= (id + [g, id]) \circ [[[i_1, i_2 \circ i_2] \circ out \circ f, i_2 \circ i_1] \circ out, (id + i_2) \circ out].
\end{aligned}$$

Thus, $[g, id] = [\![\alpha]\!]$ by uniqueness of $[\![\alpha]\!]$. It follows that indeed

$$g = [g, id] \circ i_1 = [\![\alpha]\!] \circ i_1 = f^*.$$

(D3) Take $\tau := [\![dstl \circ (id \times out)]\!] : X \times DY \to D(X \times Y)$, the requisite property then follows by definition. □

Lemma 4.7. *The following properties of* **D** *hold:*

1. $out \circ Df = (f + Df) \circ out$
2. $f^* = [f, (later \circ f)^*] \circ out$
3. $later \circ f^* = (later \circ f)^* = f^* \circ later$

Proof. These identities follow by use of Lemma 4.6:

1. Note that definitionally: $Df = (now \circ f)^*$ for any $f : X \to TY$. The statement is then simply a consequence of (D1) and (D2):

$$\begin{aligned}
&out \circ Df \\
&= out \circ (now \circ f)^* \\
&= [out \circ now \circ f, i_2 \circ (now \circ f)^*] \circ out && \text{(D2)} \\
&= (f + Df) \circ out. && \text{(D1)}
\end{aligned}$$

2. By uniqueness of f^* it suffices to show:

$$\begin{aligned}
&out \circ [f, (later \circ f)^*] \circ out \\
&= [out \circ f, out \circ (later \circ f)^*] \circ out \\
&= [out \circ f, [out \circ later \circ f, i_2 \circ (later \circ f)^*] \circ out] \circ out && \text{(D2)} \\
&= [out \circ f, i_2 \circ [f, (later \circ f)^*] \circ out] \circ out. && \text{(D1)}
\end{aligned}$$

3. This equational chain follows by monicity of out:

$$\begin{aligned}
&out \circ (later \circ f)^* \\
&= [out \circ later \circ f, i_2 \circ (later \circ f)^*] \circ out && \text{(D2)} \\
&= i_2 \circ [f, (later \circ f)^*] \circ out && \text{(D1)} \\
&= i_2 \circ f^* \\
&= out \circ later \circ f^* && \text{(D1)} \\
&= i_2 \circ f^* && \text{(D1)} \\
&= [out \circ f, i_2 \circ f^*] \circ i_2 \\
&= [out \circ f, i_2 \circ f^*] \circ out \circ later && \text{(D1)} \\
&= out \circ f^* \circ later. && \text{(D2)}
\end{aligned}$$

Thus, the postulated properties have been proven. □

Lemma 4.8. $\mathbf{D} := (D, now, (-)^*)$ *is a Kleisli triple.*

Proof. We will now use the properties proven in Lemma 4.6 to prove the Kleisli triple laws:

(K1) By uniqueness of now^* it suffices to show that $out \circ id = [out \circ now, i_2 \circ id] \circ out$ which instantly follows by (D1).

(K2) Let $f : X \to DY$. We proceed by monicity of out:

$$\begin{aligned}
&out \circ f^* \circ now \\
&= [out \circ f, i_2 \circ f^*] \circ out \circ now && \text{(D2)} \\
&= [out \circ f, i_2 \circ f^*] \circ i_1 && \text{(D1)} \\
&= out \circ f.
\end{aligned}$$

(K3) Let $f : Y \to Z, g : X \to Z$ to show $f^* \circ g^* = (f^* \circ g)^*$ by uniqueness of $(f^* \circ g)^*$ it suffices to show:

$$\begin{aligned}
&out \circ f^* \circ g^* \\
&= [out \circ f, i_2 \circ f^*] \circ out \circ g^* && \text{(D2)} \\
&= [out \circ f, i_2 \circ f^*] \circ [out \circ g, i_2 \circ g^*] \circ out && \text{(D2)} \\
&= [[out \circ f, i_2 \circ f^*] \circ out \circ g, i_2 \circ f^* \circ g^*] \circ out \\
&= [out \circ f^* \circ g, i_2 \circ f^* \circ g^*] \circ out. && \text{(D2)}
\end{aligned}$$

This concludes the proof. □

Terminality of the coalgebras $(DX, out : DX \to X + DX)_{X \in |\mathscr{C}|}$ yields the following proof principle.

Remark 4.9 (Proof by coinduction)**.** Given two morphisms $f, g : X \to DY$. To show that $f = g$ it suffices to show that there exists a coalgebra structure $\alpha : X \to Y + X$ such that the following diagrams commute:

$$\begin{array}{ccc} X & \xrightarrow{\alpha} & Y + X \\ \downarrow{\scriptstyle f} & & \downarrow{\scriptstyle id+f} \\ DY & \xrightarrow{out} & Y + DY \end{array} \qquad \begin{array}{ccc} X & \xrightarrow{\alpha} & Y + X \\ \downarrow{\scriptstyle g} & & \downarrow{\scriptstyle id+g} \\ DY & \xrightarrow{out} & Y + DY \end{array}$$

Uniqueness of the coalgebra morphism $[\![\alpha]\!] : (X, \alpha) \to (DY, out)$ then results in $f = g$.

Lemma 4.10. D *is a strong monad.*

Proof. Most of the following proofs are done via coinduction (Remark 4.9). We will only give the requisite coalgebra structure. The proofs that the diagrams commute can be looked up in the Agda formalization.

First we need to show naturality of τ, i.e. we need to show that

$$\tau \circ (f \times (now \circ g)^*) = (now \circ (f \times g))^* \circ \tau$$

The coalgebra for coinduction is:

$$\begin{array}{ccccccc} X \times DY & \xrightarrow{id \times out} & X \times (Y + DY) & \xrightarrow{dstl} & X \times Y + X \times DY & \xrightarrow{f \times g + id} & A \times B + X \times DY \\ \downarrow{\scriptstyle [\![\text{-}]\!]} & & & & & & \downarrow{\scriptstyle id + [\![\text{-}]\!]} \\ D(A \times B) & & & & \xrightarrow{\qquad\qquad out \qquad\qquad} & & A \times B + D(A \times B) \end{array}$$

We write $[\![\text{-}]\!]$ to abbreviate the used coalgebra, i.e. in the previous diagram

$$[\![\text{-}]\!] = [\![(f \times g + id) \circ dstl \circ (id \times out)]\!].$$

Next we check the strength laws:

(S1) To show that $(now \circ \pi_2)^* \circ \tau = \pi_2$ we do coinduction using the following coalgebra:

$$\begin{array}{ccccccc} 1 \times DX & \xrightarrow{id \times out} & 1 \times X + DX & \xrightarrow{dstl} & 1 \times X + 1 \times DX & \xrightarrow{\pi_2 + id} & X + 1 \times DX \\ \downarrow{\scriptstyle [\![\text{-}]\!]} & & & & & & \downarrow{\scriptstyle id + [\![\text{-}]\!]} \\ DX & & & & \xrightarrow{\qquad\qquad out \qquad\qquad} & & X + DX \end{array}$$

(S2) We don't need coinduction to show $\tau \circ (id \times now) = now$, but we will first need to establish

$$\tau \circ (id \times out^{-1}) = out^{-1} \circ (id + \tau) \circ dstl, \qquad (*)$$

which is a direct consequence of (D3). With this we are done by

$$\begin{aligned} & \tau \circ (id \times now) \\ &= \tau \circ (id \times out^{-1}) \circ (id \times i_1) \\ &= out^{-1} \circ (id + \tau) \circ dstl \circ (id \times i_1) \qquad (*) \\ &= now. \end{aligned}$$

(S3) We need to check $\tau^* \circ \tau = \tau \circ (id \times id^*)$, the coalgebra for coinduction is:

$$\begin{array}{ccccc} X \times DDY & \xrightarrow{id \times out} & X \times (DY + DDY) & \xrightarrow{dstl} & X \times DY + X \times DDY \\ & & & & \downarrow{\scriptstyle [(id + (id \times now)) \circ dstl \circ (id \times out), i_2]} \\ \downarrow{\scriptstyle [\![\text{-}]\!]} & & & & X \times Y + X \times DDY \\ & & & & \downarrow{\scriptstyle id + [\![\text{-}]\!]} \\ D(X \times Y) & & & \xrightarrow{\qquad out \qquad} & X \times Y + D(X \times Y) \end{array}$$

(S4) To show $D\alpha \circ \tau = \tau \circ (id \times \tau) \circ \alpha$ by coinduction we take the coalgebra:

$$\begin{array}{ccccc}
(X \times Y) \times DZ & \xrightarrow{id \times out} & (X \times Y) \times (Z + DZ) & \xrightarrow{dstl} & (X \times Y) \times Z + (X \times Y) \times DZ \\
\Big\downarrow{\scriptstyle [\![\text{-}]\!]} & & & & \Big\downarrow{\scriptstyle \langle \pi_1 \circ \pi_1, \langle \pi_2 \circ \pi_1, \pi_2 \rangle \rangle + id} \\
 & & & & X \times Y \times Z + (X \times Y) \times DZ \\
 & & & & \Big\downarrow{\scriptstyle id + [\![\text{-}]\!]} \\
D(X \times Y \times Z) & & \xrightarrow{\qquad\qquad out \qquad\qquad} & & X \times Y \times Z + D(X \times Y \times Z)
\end{array}$$

Thus, it has been shown that **D** is a strong monad. □

To prove that **D** is commutative we will use another proof principle previously called the *Solution Theorem* [10] or *Parametric Corecursion* [8]. In our setting this takes the following form.

Definition 4.11. We call a morphism $g : X \to D(Y + X)$ *guarded* if there exists a morphism $h : X \to Y + D(Y + X)$ such that the following diagram commutes:

$$\begin{array}{ccc}
X & \xrightarrow{\qquad g \qquad} & D(Y + X) \\
\Big\downarrow{\scriptstyle h} & & \Big\downarrow{\scriptstyle out} \\
Y + D(Y + X) & \xrightarrow[i_1 + id]{} & (Y + X) + D(Y + X)
\end{array}$$

Corollary 4.12 (Solution Theorem)**.** *Let $g : X \to D(Y + X)$ be guarded. Solutions of g are unique, i.e. given two morphisms $f, i : X \to DY$ then $f = [now, f]^* \circ g$ and $i = [now, i]^* \circ g$ already implies $f = i$.*

Proof. Let $g : X \to D(Y + X)$ be guarded by $h : X \to Y + D(Y + X)$ and $f, i : X \to DY$ be solutions of g. It suffices to show $[now, f]^* = [now, i]^*$, because then follows that

$$f = [now, f]^* \circ g = [now, i]^* \circ g = i.$$

This is proven by coinduction using

$$\begin{array}{ccccc}
D(Y + X) & \xrightarrow{out} & (Y + X) + D(Y + X) & \xrightarrow{[[i_1, h], i_2]} & Y + D(Y + X) \\
\Big\downarrow{\scriptstyle [\![\text{-}]\!]} & & & & \Big\downarrow{\scriptstyle id + [\![\text{-}]\!]} \\
DY & & \xrightarrow{\qquad\qquad out \qquad\qquad} & & Y + DY
\end{array}$$

which concludes the proof. □

Let us record some facts that we will use to prove commutativity of **D**:

Corollary 4.13. *These properties of τ and σ hold:*

$$\begin{aligned}
out \circ \tau &= (id + \tau) \circ dstl \circ (id \times out) && (\tau_1) \\
out \circ \sigma &= (id + \sigma) \circ dstr \circ (out \times id) && (\sigma_1) \\
\tau \circ (id \times out^{-1}) &= out^{-1} \circ (id + \tau) \circ dstl && (\tau_2) \\
\sigma \circ (out^{-1} \times id) &= out^{-1} \circ (id + \sigma) \circ dstr && (\sigma_2)
\end{aligned}$$

Proof. (τ_1) This is just (D3) restated.

(σ_1) Indeed, by use of (τ_1)

$$\begin{aligned}
& out \circ \sigma \\
&= out \circ Dswap \circ \tau \circ swap \\
&= (swap + Dswap) \circ out \circ \tau \circ swap \\
&= (swap + Dswap) \circ (id + \tau) \circ dstl \circ (id \times out) \circ swap && (\tau_1) \\
&= (swap + Dswap) \circ (id + \tau) \circ dstl \circ swap \circ (out \times id) \\
&= (swap + Dswap) \circ (id + \tau) \circ (swap + swap) \circ dstr \circ (out \times id) \\
&= (id + \sigma) \circ dstr \circ (out \times id).
\end{aligned}$$

(τ_2) By monicity of *out*:

$$\begin{aligned}
& out \circ \tau \circ (id \times out^{-1}) \\
&= (id + \tau) \circ dstl \circ (id \times out) \circ (id \times out^{-1}) && (\tau_1) \\
&= (id + \tau) \circ dstl.
\end{aligned}$$

(σ_2) Again, by monicity of *out*:

$$\begin{aligned}
& out \circ \sigma \circ (out^{-1} \times id) \\
&= (id + \sigma) \circ dstr \circ (out \times id) \circ (out^{-1} \times id) && (\sigma_1) \\
&= (id + \sigma) \circ dstr. && \square
\end{aligned}$$

Theorem 4.14. D *is commutative.*

Proof. Using Corollary 4.12 it suffices to show that both $\tau^* \circ \sigma$ and $\sigma^* \circ \tau$ are solutions of some guarded morphism g.

Let $w := (dstr + dstr) \circ dstl \circ (out \times out)$ and take

$$g := out^{-1} \circ [i_1 + Di_1 \circ \sigma, i_2 \circ [Di_1 \circ \tau, later \circ now \circ i_2]] \circ w.$$

Note that g is trivially guarded by $[id + Di_1 \circ \sigma, i_2 \circ [Di_1 \circ \tau, later \circ now \circ i_2]] \circ w$. It thus suffices to show that both $\tau^* \circ \sigma$ and $\sigma^* \circ \tau$ are solutions of g. Consider

$$\tau^* \circ \sigma = out^{-1} \circ [id + \sigma, i_2 \circ [\tau, later \circ \tau^* \circ \sigma]] \circ w = [now, \tau^* \circ \sigma]^* \circ g,$$

and

$$\sigma^* \circ \tau = out^{-1} \circ [id + \sigma, i_2 \circ [\tau, later \circ \sigma^* \circ \tau]] \circ w = [now, \sigma^* \circ \tau]^* \circ g.$$

The last step in both equations can be proven generally for any $f : DX \times DY \to D(X \times Y)$ using monicity of *out*:

$$\begin{aligned}
& out \circ [now, f]^* \circ out^{-1} \circ [i_1 + Di_1 \circ \sigma, i_2 \circ [Di_1 \circ \tau, later \circ now \circ i_2]] \circ w \\
&= [out \circ [now, f], i_2 \circ [now, f]^*] \circ [i_1 + Di_1 \circ \sigma, i_2 \circ [Di_1 \circ \tau, later \circ now \circ i_2]] \circ w && (D2) \\
&= [id + \sigma, i_2 \circ [now, f]^* \circ [Di_1 \circ \tau, later \circ now \circ i_2]] \circ w && (D1) \\
&= [id + \sigma, i_2 \circ [\tau, [now, f]^* \circ later \circ now \circ i_2]] \circ w \\
&= [id + \sigma, i_2 \circ [\tau, [later \circ now, later \circ f]^* \circ now \circ i_2]] \circ w \\
&= [id + \sigma, i_2 \circ [\tau, later \circ f]] \circ w.
\end{aligned}$$

Let us now check the first step in the equation for $\sigma^* \circ \tau$, the same step for $\tau^* \circ \sigma$ is then symmetric. Again, we proceed by monicity of out, which yields

$$
\begin{aligned}
& out \circ \sigma^* \circ \tau \\
&= [out \circ \sigma, i_2 \circ \sigma^*] \circ out \circ \tau && \text{(D2)} \\
&= [out \circ \sigma, i_2 \circ \sigma^*] \circ (id + \tau) \circ dstl \circ (id \times out) && \text{(D3)} \\
&= [(id + \sigma) \circ dstr \circ (out \times id), i_2 \circ \sigma^* \circ \tau] \circ dstl \circ (id \times out) && (\sigma_1) \\
&= [(id + \sigma) \circ dstr \circ (out \times id), i_2 \circ \sigma^* \circ \tau] \circ ((out^{-1} \times id) + (out^{-1} \times id)) \circ dstl \circ (out \times out) \\
&= [(id + \sigma) \circ dstr, i_2 \circ \sigma^* \circ \tau \circ (out^{-1} \times id)] \circ dstl \circ (out \times out) \\
&= [(id + \sigma) \circ dstr, i_2 \circ \sigma^* \circ D(out^{-1} \times id) \circ \tau] \circ dstl \circ (out \times out) \\
&= [(id + \sigma) \circ dstr, i_2 \circ (\sigma \times (out^{-1} \times id))^* \circ \tau] \circ dstl \circ (out \times out) \\
&= [(id + \sigma) \circ dstr, i_2 \circ (out^{-1} \circ (id + \sigma) \circ dstr)^* \circ \tau] \circ dstl \circ (out \times out) && (\sigma_2) \\
&= [(id + \sigma) \circ dstr, i_2 \circ (out^{-1} \circ (id + \sigma))^* \circ Ddstr \circ \tau] \circ dstl \circ (out \times out) \\
&= [(id + \sigma) \circ dstr, i_2 \circ (out^{-1} \circ (id + \sigma))^* \circ [Di_1 \circ \tau, Di_2 \circ \tau] \circ dstr] \circ dstl \circ (out \times out) && (*) \\
&= [(id + \sigma), i_2 \circ (out^{-1} \circ (id + \sigma))^* \circ [Di_1 \circ \tau, Di_2 \circ \tau]] \circ (dstr + dstr) \circ dstl \circ (out \times out) \\
&= [(id + \sigma), i_2 \circ [(out^{-1} \circ i_1)^* \circ \tau, (out^{-1} \circ i_2 \circ \sigma)^* \circ \tau]] \circ w \\
&= [(id + \sigma), i_2 \circ [\tau, (later \circ \sigma)^* \circ \tau]] \circ w && \text{(K1)} \\
&= [(id + \sigma), i_2 \circ [\tau, later \circ \sigma^* \circ \tau]] \circ w,
\end{aligned}
$$

where

$$Ddstr \circ \tau = [Di_1 \circ \tau, Di_2 \circ \tau] \circ dstr \qquad (*)$$

indeed follows by epicness of $dstr^{-1}$:

$$
\begin{aligned}
& Ddstr \circ \tau \circ dstr^{-1} \\
&= [Ddstr \circ \tau \circ (i_1 \times id), Ddstr \circ \tau \circ (i_2 \times id)] \\
&= [Ddstr \circ D(i_1 \times id) \circ \tau, Ddstr \circ D(i_2 \times id) \circ \tau] \\
&= [Di_1 \circ \tau, Di_2 \circ \tau]. && \square
\end{aligned}
$$

We have now seen that $\mathbf{D}$ is strong and commutative, however it is not an equational lifting monad, since besides modeling non-termination, the delay monad also counts the execution time of a computation. This is a result of the too intensional notion of equality that this monad comes with.

In Chapter 6 we will see a way to remedy this: taking the quotient of the delay monad where execution time is ignored. This will then yield an equational lifting monad on the category of setoids. However, in a general setting it is generally believed to be impossible to define a monad structure on this quotient. Chapman et al. [16] have identified the axiom of countable choice (which crucially holds in the category of setoids) as a sufficient requirement for defining a monad structure on the quotient of $\mathbf{D}$.

5 Iteration Algebras and Monads

In this chapter we will draw on the inherent connection between partiality and iteration to establish a partiality monad in a general setting without axioms by utilizing previous research on iteration theories.

5.1 Elgot Algebras

Recall the following notion from [13], previously called *complete Elgot algebra*.

Definition 5.1 (Guarded Elgot Algebra)**.** Given a functor $H : \mathscr{C} \to \mathscr{C}$, a *(H-)guarded Elgot algebra* consists of:

- An object $A \in |\mathscr{C}|$,
- a H-algebra structure $a : H\ A \to A$,
- and for every $f : X \to A + HX$ an *iteration* $f^\sharp : X \to A$, satisfying the following axioms:
  - **Fixpoint**: $f^\sharp = [id, a \circ H(f^\sharp)] \circ f$
    for any $f : X \to A + HX$,
  - **Uniformity**: $(id + Hh) \circ f = g \circ h$ implies $f^\sharp = g^\sharp \circ h$
    for any $f : X \to A + HX, g : Y \to A + HY, h : X \to Y$,
  - **Compositionality**: $((f^\sharp + id) \circ h)^\sharp = ([(id + Hi_1) \circ f, i_2 \circ Hi_2] \circ [i_1, h])^\sharp \circ i_2$
    for any $f : X \to A + HX, h : Y \to X + HY$.

Consider an Elgot algebra over the identity functor $Id : \mathscr{C} \to \mathscr{C}$ together with the trivial Id-algebra structure $id : A \to A$. Morphisms of the form $f : X \to A + X$ can then be viewed as modeling one iteration of a loop, where $X \in |\mathscr{C}|$ is the state space and $A \in |\mathscr{C}|$ the object of values. Intuitively, in such a setting the iteration operator $(-)^\sharp$ runs such a morphism in a loop until it terminates (or diverges), thus assigning it a solution. This is what the **Fixpoint** axiom guarantees. On the other hand the **Uniformity** axiom states how to handle loop invariants and finally, the **Compositionality** axiom is the most sophisticated one, stating that compatible iterations can be combined into a single iteration with a merged state space.

The previous intuition gives rise to the following simpler definition that has been introduced in [19].

Definition 5.2 (Elgot Algebra)**.** A *(unguarded) Elgot Algebra* [19] consists of:

- An object $A \in |\mathscr{C}|$,
- and for every $f : X \to A + X$ an *iteration* $f^\sharp : X \to A$, satisfying the following axioms:
  - **Fixpoint**: $f^\sharp = [id, f^\sharp] \circ f$
    for any $f : X \to A + X$,
  - **Uniformity**: $(id + h) \circ f = g \circ h$ implies $f^\sharp = g^\sharp \circ h$
    for any $f : X \to A + X,\ g : Y \to A + Y,\ h : X \to Y$,

– **Folding**: $((f^\sharp + id) \circ h)^\sharp = [(id + i_1) \circ f, i_2 \circ h]^\sharp$
for any $f : X \to A + X,\ h : Y \to X + Y$.

Note that the **Uniformity** axiom requires an identity to be proven, before it can be applied. However, we will omit these proofs in most parts of the thesis, since they mostly follow by simple rewriting on (co)products, the full proofs can be looked up in the accompanying formalization.

Now, in this setting the simpler **Folding** axiom replaces the sophisticated **Compositionality** axiom. Indeed, for Id-guarded Elgot algebras with a trivial algebra structure, the **Folding** and **Compositionality** axioms are equivalent [19], which is partly illustrated by the following Lemma.

Lemma 5.3. *Every Elgot algebra $(A, (-)^\sharp)$ satisfies the following additional axioms*

- ***Compositionality**: $((f^\sharp + id) \circ h)^\sharp = ([(id + i_1) \circ f, i_2 \circ i_2] \circ [i_1, h])^\sharp \circ i_2$
for any $f : X \to A + X, h : Y \to X + Y$,*
- ***Stutter**: $(([h, h] + id) \circ f)^\sharp = (i_1 \circ h, [h + i_1, i_2 \circ i_2])^\sharp \circ inr$
for any $f : X \to (Y + Y) + X, h : Y \to A$,*
- ***Diamond**: $((id + \Delta) \circ f)^\sharp = ([i_1, ((id + \Delta) \circ f)^\sharp + id] \circ f)^\sharp$
for any $f : X \to A + (X + X)$.*

Proof. The proofs of the axioms build upon each other, we prove them one by one.

- **Compositionality**: First, note that **Folding** can equivalently be reformulated as

$$((f^\sharp + id) \circ h)^\sharp = [(id + i_1) \circ f, i_2 \circ h]^\sharp \circ i_2, \qquad \textbf{(Folding')}$$

since

$$\begin{aligned}
&((f^\sharp + id) \circ h)^\sharp \\
&= (f^\sharp + h)^\sharp \circ h && \textbf{(Uniformity)} \\
&= [f^\sharp, (f^\sharp + h)^\sharp \circ h] \circ i_2 \\
&= [id, (f^\sharp + h)^\sharp] \circ (f^\sharp + h) \circ i_2 \\
&= (f^\sharp + h)^\sharp \circ i_2 && \textbf{(Fixpoint)} \\
&= [(id + i_1) \circ f, i_2 \circ h]^\sharp \circ i_2. && \textbf{(Folding)}
\end{aligned}$$

Using **Folding'**, it suffices to show that

$$[(id + i_1) \circ f, i_2 \circ h]^\sharp \circ i_2 = ([(id + i_1) \circ f, i_2 \circ i_2] \circ [i_1, h])^\sharp \circ i_2.$$

Indeed,

$$\begin{aligned}
&[(id + i_1) \circ f, i_2 \circ h]^\sharp \circ i_2 \\
&= [id, [(id + i_1) \circ f, i_2 \circ h]^\sharp] \circ [(id + i_1) \circ f, i_2 \circ h] \circ i_2 && \textbf{(Fixpoint)} \\
&= [id, [(id + i_1) \circ f, i_2 \circ h]^\sharp] i_2 \circ h \\
&= [(id + i_1) \circ f, i_2 \circ h]^\sharp \circ h \\
&= [(id + i_1) \circ f, i_2 \circ h]^\sharp [i_1, h] \circ i_2 \\
&= ([(id + i_1) \circ f, i_2 \circ i_2] \circ [i_1, h])^\sharp \circ i_2. && \textbf{(Uniformity)}
\end{aligned}$$

- **Stutter**: Let us first establish

$$[h, h] = (h + i_1)^\sharp, \tag{*}$$

which follows by

$$\begin{aligned}
&(h + i_1)^\sharp \\
&= [id, (h + i_1)^\sharp] \circ (h + i_1) && (\textbf{Fixpoint}) \\
&= [h, (h + i_1)^\sharp \circ i_1] \\
&= [h, [id, (h + i_1)^\sharp] \circ (h + i_1) \circ i_1] && (\textbf{Fixpoint}) \\
&= [h, h].
\end{aligned}$$

Now we are done by

$$\begin{aligned}
&([h, h] + id) \circ f^\sharp \\
&= ((h + i_1)^\sharp + id) \circ f^\sharp && (*) \\
&= ([(id + i_1) \circ (h + i_1), i_2 \circ i_2] \circ [i_1, f])^\sharp \circ i_2 && (\textbf{Compositionality}) \\
&= ([h + i_1 \circ i_1, i_2 \circ i_2] \circ [i_1, f])^\sharp \circ (i_1 + id) \circ i_2 \\
&= [i_1 \circ h, [h + i_1, i_2 \circ i_2] \circ f]^\sharp \circ i_2. && (\textbf{Uniformity})
\end{aligned}$$

- **Diamond**: Let $h = [i_1 \circ i_1, i_2 + id] \circ f$ and $g = (id + \Delta) \circ f$.
First, note that

$$[id, g^\sharp] = [i_1, (id + i_2) \circ g]^\sharp, \tag{$*$}$$

by **Fixpoint** and **Uniformity**:

$$\begin{aligned}
&[id, g^\sharp] \\
&= [id, [id, g^\sharp] \circ g] && (\textbf{Fixpoint}) \\
&= [id, [id, [i_1, (id + i_2) \circ g]^\sharp \circ i_2] \circ g] && (\textbf{Uniformity}) \\
&= [id, [i_1, (id + i_2) \circ g]^\sharp] \circ [i_1, (id + i_2) \circ g] \\
&= [i_1, (id + i_2) \circ g]^\sharp. && (\textbf{Fixpoint})
\end{aligned}$$

It thus suffices to show that,

$$\begin{aligned}
&g^\sharp \\
&= [(id + i_1) \circ [i_1, (id + i_2) \circ g], i_2 \circ h]^\sharp \circ i_2 \\
&= ([i_1, (id + i_2) \circ g]^\sharp + h)^\sharp \circ i_2 && (\textbf{Folding}) \\
&= ([i_1, g^\sharp + id] \circ f)^\sharp.
\end{aligned}$$

Indeed,

$$
\begin{aligned}
& g^\sharp \\
&= g^\sharp \circ [id, id] \circ i_2 \\
&= [(id + i_2) \circ g, f]^\sharp \circ i_2 && \textbf{(Uniformity)} \\
&= (([id, id] + id) \circ [(i_1 + i_1) \circ g, (i_2 + id) \circ f])^\sharp \circ i_2 \\
&= [i_1, [id + i_1, i_2 \circ i_2] \circ [(i_1 + i_1) \circ g, (i_2 + id) \circ f]]^\sharp \circ i_2 \circ i_2 && \textbf{(Stutter)} \\
&= [i_1, [[i_1, i_2 \circ i_2 \circ i_1] \circ g, [i_2 \circ i_1, i_2 \circ i_2] \circ f]]^\sharp \circ i_2 \circ i_2 \\
&= [i_1, [(id + i_2 \circ i_1) \circ g, i_2 \circ \circ f]]^\sharp \circ i_2 \circ i_2 \\
&= [[i_1, (id + i_1 \circ i_2) \circ g], i_2 \circ h]^\sharp \circ [i_1 \circ i_1, i_2 + id] \circ i_2 \circ i_2 && \textbf{(Uniformity)} \\
&= [[i_1, (id + i_1 \circ i_2) \circ g], i_2 \circ h]^\sharp \circ i_2 \\
&= [(id + i_1) \circ [i_1, (id + i_2) \circ g], i_2 \circ h]^\sharp \circ i_2
\end{aligned}
$$

and

$$
\begin{aligned}
& ([i_1, g^\sharp + id] \circ f)^\sharp \\
&= ([i_1 \circ [id, g^\sharp] \circ i_1, [id, g^\sharp] \circ i_2 + id] \circ f)^\sharp \\
&= (([id, g^\sharp] + id) \circ h)^\sharp \\
&= ((([[id, g^\sharp], [id, g^\sharp]] + id) \circ (i_2 + id) \circ h)^\sharp \\
&= [i_1 \circ [id, g^\sharp], [[id, g^\sharp] + i_1, i_2 \circ i_2] \circ (i_2 + id) \circ h]^\sharp \circ i_2 && \textbf{(Stutter)} \\
&= ([id, g^\sharp] + h)^\sharp \circ i_2 \\
&= ([i_1, (id + i_2) \circ g]^\sharp + h)^\sharp \circ i_2, && (*)
\end{aligned}
$$

which concludes the proof. □

Note that in [19] it has been shown that the **Diamond** axiom implies **Compositionality**, yielding another definition of Elgot algebras only requiring the **Fixpoint**, **Uniformity** and **Diamond** axioms.

Let us now consider morphisms that are coherent with respect to the iteration operator. A special case being morphisms between Elgot algebras.

Definition 5.4 (Iteration Preserving Morphisms)**.** Let $(A, (-)^{\sharp_a}), (B, (-)^{\sharp_b})$ be two Elgot algebras.

A morphism $f : X \times A \to B$ is called *right iteration preserving* if

$$f \circ (id \times h^{\sharp_a}) = ((f + id) \circ dstl \circ (id \times h))^{\sharp_b}$$

for any $h : Y \to A + Y$.

Symmetrically a morphism $g : A \times X \to B$ is called *left iteration preserving* if

$$f \circ (h^{\sharp_a} \times id) = ((f + id) \circ dstr \circ (h \times id))^{\sharp_b}$$

for any $h : Y \to A + Y$.

Let us also consider the special case where $X = 1$. A morphism $f : A \to B$ is called *iteration preserving* if

$$f \circ h^{\sharp_a} = ((f + id) \circ h)^{\sharp_b}$$

for any $h : Y \to A + Y$.

We will now study the category of Elgot algebras and iteration preserving morphisms that we call *ElgotAlgs*($\mathscr{C}$). Let us introduce notation for morphisms between Elgot algebras: we denote an Elgot algebra morphism $f : (A, (-)^{\sharp_a}) \to (B, (-)^{\sharp_b})$ as $f : A \hookrightarrow B$, where we omit stating the iteration operator.

Lemma 5.5. *ElgotAlgs*($\mathscr{C}$) *is a category.*

Proof. It suffices to show that the identity morphism in $\mathscr{C}$ is iteration preserving and the composite of two iteration preserving morphisms is also iteration preserving.

Let A, B and C be Elgot algebras. The identity trivially satisfies

$$id \circ f^{\sharp_a} = f^{\sharp_a} = ((id + id) \circ f)^{\sharp_a}$$

for any $f : X \to A + X$. Given two iteration preserving morphisms $f : B \hookrightarrow C, g : A \hookrightarrow B$, the composite is iteration preserving since

$$\begin{aligned} & f \circ g \circ h^{\sharp_a} \\ = {} & f \circ ((g + id) \circ h)^{\sharp_b} \\ = {} & ((f + id) \circ (g + id) \circ h)^{\sharp_c} \\ = {} & ((f \circ g + id) \circ h)^{\sharp_c} \end{aligned}$$

for any $h : X \to A + X$. □

Products and exponentials of Elgot algebras can be formed in a canonical way, which is illustrated by the following two Lemmas.

Lemma 5.6. *If $\mathscr{C}$ is a Cartesian category, so is ElgotAlgs*($\mathscr{C}$).

Proof. Let 1 be the terminal object of $\mathscr{C}$. Given $f : X \to 1 + X$ we define the iteration $f^\sharp = ! : X \to 1$ as the unique morphism into the terminal object. The Elgot algebra laws follow instantly by uniqueness of ! and $(1, !)$ is the terminal Elgot algebra since for every Elgot algebra A the morphism $! : A \to 1$ extends to a morphism between Elgot algebras by uniqueness.

Let $A, B \in |\mathit{ElgotAlgs}(\mathscr{C})|$ and $A \times B$ be the product of A and B in $\mathscr{C}$. Given $f : X \to (A \times B) + X$ we define the iteration as:

$$f^\sharp = \langle ((\pi_1 + id) \circ f)^{\sharp_a}, ((\pi_2 + id) \circ f)^{\sharp_b} \rangle : X \to A \times B$$

Now, we show that $A \times B$ indeed constitutes an Elgot algebra:

- **Fixpoint**: Let $f : X \to (A \times B) + X$. The requisite equation follows by the fixpoint identities of $((\pi_1 + id) \circ f)^{\sharp_a}$ and $((\pi_2 + id) \circ f)^{\sharp_b}$:

$$\begin{aligned}
&\langle ((\pi_1 + id) \circ f)^{\sharp_a}, ((\pi_2 + id) \circ f)^{\sharp_b} \rangle \\
&= \langle [id, ((\pi_1 + id) \circ f)^{\sharp_a}] \circ (\pi_1 + id) \circ f \\
&\quad , [id, ((\pi_2 + id) \circ f)^{\sharp_b}] \circ (\pi_2 + id) \circ f \rangle && \textbf{(Fixpoint)} \\
&= \langle [\pi_1, ((\pi_1 + id) \circ f)^{\sharp_a}] \circ f, [\pi_2, ((\pi_2 + id) \circ f)^{\sharp_b}] \circ f \rangle \\
&= \langle [\pi_1, ((\pi_1 + id) \circ f)^{\sharp_a}], [\pi_2, ((\pi_2 + id) \circ f)^{\sharp_b}] \rangle \circ f \\
&= [\langle \pi_1, \pi_2 \rangle, \langle ((\pi_1 + id) \circ f)^{\sharp_a}, ((\pi_2 + id) \circ f)^{\sharp_b} \rangle] \circ f \\
&= [id, \langle ((\pi_1 + id) \circ f)^{\sharp_a}, ((\pi_2 + id) \circ f)^{\sharp_b} \rangle] \circ f
\end{aligned}$$

- **Uniformity**: Let $f : X \to (A \times B) + X, g : Y \to (A \times B) + Y, h : X \to Y$ and $(id + h) \circ f = g \circ h$. Note that this implies:

$$\begin{aligned}
(id + h) \circ (\pi_1 + id) \circ f &= (\pi_1 + id) \circ g \circ h \\
(id + h) \circ (\pi_2 + id) \circ f &= (\pi_2 + id) \circ g \circ h
\end{aligned}$$

  Then, **Uniformity** of $(-)^{\sharp_a}$ and $(-)^{\sharp_b}$ with the previous two equations yields:

$$\begin{aligned}
\langle ((\pi_1 + id) \circ f)^{\sharp_a}, ((\pi_1 + id) \circ f)^{\sharp_a} \rangle &= ((\pi_1 + id) \circ g)^{\sharp_a} \circ h \\
\langle ((\pi_2 + id) \circ f)^{\sharp_b}, ((\pi_2 + id) \circ f)^{\sharp_b} \rangle &= ((\pi_2 + id) \circ g)^{\sharp_b} \circ h
\end{aligned}$$

  This concludes the proof of:

$$\langle ((\pi_1 + id) \circ f)^{\sharp_a}, ((\pi_2 + id) \circ f)^{\sharp_b} \rangle = \langle ((\pi_1 + id) \circ g)^{\sharp_a}, ((\pi_2 + id) \circ g)^{\sharp_b} \rangle \circ h$$

- **Folding**: Let $f : X \to (A \times B) + X, h : Y \to X + Y$. We need to show:

$$\begin{aligned}
&\langle ((\pi_1 + id) \circ (\langle ((\pi_1 + id) \circ f)^{\sharp_a}, ((\pi_2 + id) \circ f)^{\sharp_b} \rangle + h))^{\sharp_a} \\
&\quad , ((\pi_2 + id) \circ (\langle ((\pi_1 + id) \circ f)^{\sharp_a}, ((\pi_2 + id) \circ f)^{\sharp_b} \rangle + h))^{\sharp_b} \rangle \\
&= \langle (\pi_1 + id) \circ [(id + i_1) \circ f, i_2 \circ h]^{\sharp_a}, (\pi_2 + id) \circ [(id + i_1) \circ f, i_2 \circ h]^{\sharp_b} \rangle
\end{aligned}$$

  Indeed, the folding laws for $(-)^{\sharp_a}$ and $(-)^{\sharp_b}$ imply

$$\begin{aligned}
&((\pi_1 + id) \circ (\langle ((\pi_1 + id) \circ f)^{\sharp_a}, ((\pi_2 + id) \circ f)^{\sharp_b} \rangle + h))^{\sharp_a} \\
&= (((\pi_1 + id) \circ f)^{\sharp_a} + h)^{\sharp_a} \\
&= [(id + i_1) \circ (\pi_1 + id) \circ f, i_2 \circ h]^{\sharp_a} && \textbf{(Folding)} \\
&= (\pi_1 + id) \circ [(id + i_1) \circ f, i_2 \circ h]^{\sharp_a}
\end{aligned}$$

  and

$$\begin{aligned}
&((\pi_2 + id) \circ (\langle ((\pi_1 + id) \circ f)^{\sharp_a}, ((\pi_2 + id) \circ f)^{\sharp_b} \rangle + h))^{\sharp_b} \\
&= (((\pi_2 + id) \circ f)^{\sharp_b} + h)^{\sharp_b} \\
&= [(id + i_1) \circ (\pi_2 + id) \circ f, i_2 \circ h]^{\sharp_b} && \textbf{(Folding)} \\
&= (\pi_2 + id) \circ [(id + i_1) \circ f, i_2 \circ h]^{\sharp_b}
\end{aligned}$$

  which concludes the proof of the folding law.

The product diagram of $A \times B$ in $\mathscr{C}$ then also holds in *ElgotAlgs*($\mathscr{C}$), we just have to check that the projections are iteration preserving, which follows instantly, and that the unique morphism $\langle f, g \rangle$ is iteration preserving for any $f : C \hookrightarrow A, g : C \to B$ where $C \in |\mathit{ElgotAlgs}(\mathscr{C})|$.

Let $h : X \to C + X$. We use the fact that f and g are iteration preserving to show:

$$\begin{aligned}
&\langle f, g \rangle \circ (h^{\sharp_c}) \\
&= \langle f \circ (h^{\sharp_c}), g \circ (h^{\sharp_c}) \rangle \\
&= \langle ((f + id) \circ h)^{\sharp_a}, ((g + id) \circ h)^{\sharp_b} \rangle \\
&= \langle ((\pi_1 + id) \circ (\langle f, g \rangle + id) \circ h)^{\sharp_a}, ((\pi_1 + id) \circ (\langle f, g \rangle + id) \circ h)^{\sharp_b} \rangle
\end{aligned}$$

Which confirms that $\langle f, g \rangle$ is indeed iteration preserving. Thus, it follows that $A \times B$ extends to a product in *ElgotAlgs*($\mathscr{C}$) and therefore *ElgotAlgs*($\mathscr{C}$) is Cartesian, if $\mathscr{C}$ is Cartesian. □

Lemma 5.7. *Given $X \in |\mathscr{C}|$ and $A \in |\mathit{ElgotAlgs}(\mathscr{C})|$. The exponential X^A (if it exists) can be equipped with an Elgot algebra structure.*

Proof. Take $f^\sharp = \mathit{curry}(((\mathit{eval} + id) \circ \mathit{dstr} \circ (f \times id))^{\sharp_a})$ as the iteration of some $f : Z \to A^X + Z$.

- **Fixpoint**: Let $f : Y \to X^A + Y$. We need to show that $f^\sharp = [id, f^\sharp] \circ f$, which follows by uniqueness of

  $$f^\sharp = \mathit{curry}\ (((\mathit{eval} + id) \circ \mathit{dstr} \circ (f \times id))^{\sharp_a})$$

  and

  $$\begin{aligned}
  &\mathit{eval} \circ ([id, \mathit{curry}(((\mathit{eval} + id) \circ \mathit{dstr} \circ (f \times id))^{\sharp_a})] \circ f \times id) \\
  &= \mathit{eval} \circ [id, \mathit{curry}(((\mathit{eval} + id) \circ \mathit{dstr} \circ (f \times id))^{\sharp_a}) \times id] \circ \mathit{dstr} \circ (f \times id) && (*) \\
  &= [\mathit{eval}, \mathit{eval} \circ (\mathit{curry}(((\mathit{eval} + id) \circ \mathit{dstr} \circ (f \times id))^{\sharp_a}) \times id)] \circ \mathit{dstr} \circ (f \times id) \\
  &= [\mathit{eval}, ((\mathit{eval} + id) \circ \mathit{dstr} \circ (f \times id))^{\sharp_a}] \circ \mathit{dstr} \circ (f \times id) \\
  &= [id, ((\mathit{eval} + id) \circ \mathit{dstr} \circ (f \times id))^{\sharp_a}] \circ (\mathit{eval} + id) \circ \mathit{dstr} \circ (f \times id) \\
  &= ((\mathit{eval} + id) \circ \mathit{dstr} \circ (f \times id))^{\sharp_a}, && (\textbf{Fixpoint})
  \end{aligned}$$

  where

  $$\begin{aligned}
  &[id, \mathit{curry}(((\mathit{eval} + id) \circ \mathit{dstr} \circ (f \times id))^{\sharp_a})] \times id \\
  &= [id, \mathit{curry}(((\mathit{eval} + id) \circ \mathit{dstr} \circ (f \times id))^{\sharp_a}) \times id] \circ \mathit{dstr} && (*)
  \end{aligned}$$

  follows by post-composing with π_1 and π_2, indeed:

  $$\begin{aligned}
  &\pi_1 \circ [id, \mathit{curry}(((\mathit{eval} + id) \circ \mathit{dstr} \circ (f \times id))^{\sharp_a}) \times id] \circ \mathit{dstr} \\
  &= [\pi_1, \pi_1 \circ (\mathit{curry}(((\mathit{eval} + id) \circ \mathit{dstr} \circ (f \times id))^{\sharp_a}) \times id)] \circ \mathit{dstr} \\
  &= [\pi_1, \mathit{curry}(((\mathit{eval} + id) \circ \mathit{dstr} \circ (f \times id))^{\sharp_a}) \circ \pi_1] \circ \mathit{dstr} \\
  &= [id, \mathit{curry}(((\mathit{eval} + id) \circ \mathit{dstr} \circ (f \times id))^{\sharp_a})] \circ (\pi_1 + \pi_1) \circ \mathit{dstr} \\
  &= [id, \mathit{curry}(((\mathit{eval} + id) \circ \mathit{dstr} \circ (f \times id))^{\sharp_a})] \circ \pi_1,
  \end{aligned}$$

and

$$\begin{aligned}
&\pi_2 \circ [id, curry(((eval + id) \circ dstr \circ (f \times id))^{\sharp_a}) \times id] \circ dstr \\
&= [\pi_2, \pi_2 \circ (curry(((eval + id) \circ dstr \circ (f \times id))^{\sharp_a}) \times id)] \circ dstr \\
&= [\pi_2, \pi_2] \circ dstr \\
&= \pi_2.
\end{aligned}$$

- **Uniformity**: Let $f : Y \to X^A + Y, g : Z \to X^A + Z, h : Y \to Z$ and $(id + h) \circ f = g \circ h$. Again, by uniqueness of $f^\sharp = curry(((eval + id) \circ dstr \circ (f \times id))^{\sharp_a})$ it suffices to show:

$$\begin{aligned}
&((eval + id) \circ dstr \circ (f \times id))^{\sharp_a} \\
&= ((eval + id) \circ dstr \circ (g \times id))^{\sharp_a} \circ (h \times id) && \textbf{(Uniformity)} \\
&= eval \circ (((eval + id) \circ dstr \circ (g \times id))^{\sharp_a} \times id) \circ (h \times id) \\
&= eval \circ (((eval + id) \circ dstr \circ (g \times id))^{\sharp_a} \circ h \times id).
\end{aligned}$$

  Note that the application of **Uniformity** requires:

$$\begin{aligned}
&(id + (h \times id)) \circ (eval + id) \circ dstr \circ (f \times id) \\
&= (eval + id) \circ (id + (h \times id)) \circ dstr \circ (f \times id) \\
&= (eval + id) \circ dstr \circ (id \times h) \circ (id \times id) \circ (f \times id) \\
&= (eval + id) \circ dstr \circ (g \times id) \circ (h \times id).
\end{aligned}$$

- **Folding**: Let $f : Y \to X^A + Y, h : Y \to Z$. We need to show that

$$\begin{aligned}
&(f^\sharp + h)^\sharp \\
&= curry(((eval + id) \circ dstr \circ ((curry(((eval + id) \circ dstr \circ (f \times id))^{\sharp_a}) + h) \times id))^{\sharp_a}) \\
&= curry(((eval + id) \circ dstr \circ ([(id + i_1) \circ f, i_2 \circ h] \times id))^{\sharp_a}) \\
&= [(id + i_1) \circ f, i_2 \circ h]^\sharp.
\end{aligned}$$

  Indeed, we are done by

$$\begin{aligned}
&((eval + id) \circ dstr \circ ((curry(((eval + id) \circ dstr \circ (f \times id))^{\sharp_a}) + h) \times id))^{\sharp_a} \\
&= ((eval + id) \circ ((curry(((eval + id) \circ dstr \circ (f \times id))^{\sharp_a}) \times id) + (h \times id)) \circ dstr)^{\sharp_a} \\
&= ((eval \circ (curry(((eval + id) \circ dstr \circ (f \times id))^{\sharp_a}) \times id) + (h \times id)) \circ dstr)^{\sharp_a} \\
&= ((((eval + id) \circ dstr \circ (f \times id))^{\sharp_a} + (h \times id)) \circ dstr)^{\sharp_a} \\
&= ((((eval + id) \circ dstr \circ (f \times id))^{\sharp_a} + dstr \circ (h \times id)))^{\sharp_a} \circ dstr && \textbf{(Uniformity)} \\
&= [(id + i_1) \circ (eval + id) \circ dstr \circ (f \times id), i_2 \circ dstr \circ (h \times id)] \circ dstr && \textbf{(Folding)} \\
&= ((eval + id) \circ dstr \circ ([(id + i_1) \circ f, i_2 \circ h] \times id))^{\sharp_a}, && \textbf{(Uniformity)}
\end{aligned}$$

  where the identity that is required for the second application of **Uniformity** follows by epicness of $dstr^{-1}$. □

5.2 The Initial (Strong) Pre-Elgot Monad

In this section we will study the monad that arises from existence of all free Elgot algebras. We will show that this is an equational lifting monad and also the initial strong pre-Elgot monad. Starting in this section we will now omit indices of the iteration operator of Elgot algebras for the sake of readability.

Let us first recall the following notion that was introduced in [14] and reformulated in [19].

Definition 5.8 (Elgot Monad)**.** An Elgot monad consists of

- A monad $\mathbf{T}$,
- for every $f : X \to T(Y + X)$ an iteration $f^\dagger : X \to TY$ satisfying:
  - **Fixpoint**: $f^\dagger = [\eta, f^\dagger]^* \circ f$
    for any $f : X \to T(Y + X)$,
  - **Uniformity**: $f \circ h = T(id + h)$ implies $f^\dagger \circ g = g^\dagger$
    for any $f : X \to T(Y + X), g : Z \to T(Y + Z), h : Z \to X$,
  - **Naturality**: $g^* \circ f^\dagger = ([Ti_1 \circ g, \eta \circ i_2]^* \circ f)^\dagger$
    for any $f : X \to T(Y + X), g : Y \to TZ$,
  - **Codiagonal**: $f^{\dagger\dagger} = (T[id, i_2] \circ f)^\dagger$
    for any $f : X \to T((Y + X) + X)$.

If the monad $\mathbf{T}$ is strong with strength τ and $\tau \circ (id \times f^\dagger) = (Tdstl \circ \tau \circ (id \times f))^\dagger$ for any $f : X \to T(Y + X)$, then $\mathbf{T}$ is a strong Elgot monad.

We regard Elgot monads as minimal semantic structures for interpreting side-effecting while loops, as has been argued in [17], [18]. The following notion has been introduced in [19] as a weaker approximation of the notion of Elgot monad, using less sophisticated axioms.

Definition 5.9 (Pre-Elgot Monad)**.** A monad $\mathbf{T}$ is called pre-Elgot if every TX extends to an Elgot algebra such that for every $f : X \to TY$ the Kleisli lifting $f^* : TX \to TY$ is iteration preserving.

If the monad $\mathbf{T}$ is additionally strong and the strength τ is right iteration preserving we call $\mathbf{T}$ strong pre-Elgot.

(Strong) pre-Elgot monads form a subcategory of $\mathit{Monads}(\mathscr{C})$ where objects are (strong) pre-Elgot monads and morphisms between pre-Elgot monads are natural transformations α as in Definition 2.13 such that additionally each α_X is iteration preserving. Similarly, morphisms between strong pre-Elgot monads are natural transformations α as in Definition 2.19 such that each α_X is iteration preserving. We call these categories $\mathit{PreElgot}(\mathscr{C})$ and $\mathit{StrongPreElgot}(\mathscr{C})$ respectively.

Lemma 5.10. *$\mathit{PreElgot}(\mathscr{C})$ and $\mathit{StrongPreElgot}(\mathscr{C})$ are categories.*

Proof. Since $\mathit{PreElgot}(\mathscr{C})$ and $\mathit{StrongPreElgot}(\mathscr{C})$ are subcategories of the previously defined categories $\mathit{Monads}(\mathscr{C})$ and $\mathit{StrongMonads}(\mathscr{C})$ respectively, it suffices to show that the components of the identity natural transformation are iteration preserving and that the component wise composition of two pre-Elgot monad morphisms is iteration preserving. This has already been shown in Lemma 5.5. □

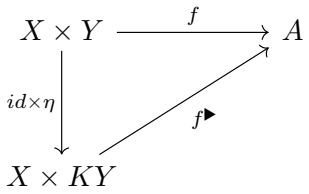
Assuming a form of the axiom of countable choice it has been proven in [19] that the initial pre-Elgot monad and the initial Elgot monad coincide, thus closing the expressivity gap in such a setting. However, it is believed to be impossible to close this gap in a general setting.

Proposition 5.11. *Existence of all free Elgot algebras yields a monad that we call* **K**.

Proof. This is a direct consequence of Proposition 2.22. □

We will need a notion of stability for **K** to make progress, since we do not assume $\mathscr{C}$ to be Cartesian closed.

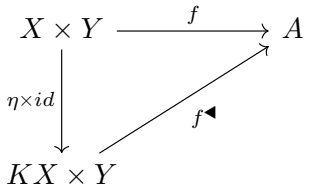
Definition 5.12 (Right-Stable Free Elgot Algebra)**.** Let KY be a free Elgot algebra on $Y \in |\mathscr{C}|$. We call KY *right-stable* if for every $A \in \mathit{ElgotAlgs}(\mathscr{C}), X \in |\mathscr{C}|$, and $f : X \times Y \to A$ there exists a unique right iteration preserving $f^{\blacktriangleright} : X \times KY \to A$ such that



commutes.

A symmetrical variant of the previous definition is sometimes useful.

Definition 5.13 (Left-Stable Free Elgot Algebra)**.** Let KY be a free Elgot algebra on $Y \in |\mathscr{C}|$. We call KY *left-stable* if for every $A \in \mathit{ElgotAlgs}(\mathscr{C}), X \in |\mathscr{C}|$, and $f : Y \times X \to A$ there exists a unique left iteration preserving $f^{\blacktriangleleft} : KY \times X \to A$ such that



commutes.

Lemma 5.14. *Definitions 5.12 and 5.13 are equivalent in the sense that they imply each other.*

Proof. Let KY be a left stable free Elgot algebra on $Y \in |\mathscr{C}|$. Furthermore, let A be an Elgot algebra and $X \in |\mathscr{C}|, f : Y \times X \to A$.

We take $f^{\blacktriangleright} := (f \circ \mathit{swap})^{\blacktriangleleft} \circ \mathit{swap}$, which is indeed right iteration preserving, since

$$
\begin{aligned}
& f^{\blacktriangleright} \circ (id \times h^{\sharp}) \\
&= (f \circ \mathit{swap})^{\blacktriangleleft} \circ \mathit{swap} \circ (id \times h^{\sharp}) \\
&= (f \circ \mathit{swap})^{\blacktriangleleft} \circ (h^{\sharp} \times id) \circ \mathit{swap} \\
&= (((f \circ \mathit{swap})^{\blacktriangleleft} + id) \circ \mathit{dstr} \circ (id \times h))^{\sharp} \circ \mathit{swap} \\
&= (((f \circ \mathit{swap})^{\blacktriangleleft} \circ \mathit{swap} + id) \circ \mathit{dstl} \circ (id \times h))^{\sharp} && (\textbf{Uniformity}) \\
&= ((f^{\blacktriangleright} + id) \circ \mathit{dstl} \circ (id \times h))^{\sharp},
\end{aligned}
$$

for any $Z \in |\mathscr{C}|, h : Z \to KY + Z$.

The requisite diagram commutes, since

$$\begin{aligned} & f^{\blacktriangleright} \circ (id \times \eta) \\ = {} & (f \circ swap)^{\blacktriangleleft} \circ swap \circ (id \times \eta) \\ = {} & (f \circ swap)^{\blacktriangleleft} \circ (\eta \times id) \circ swap \\ = {} & f \circ swap \circ swap \\ = {} & f. \end{aligned}$$

Finally, let us check uniqueness of $f^{\blacktriangleright} = (f \circ swap)^{\blacktriangleleft} \circ swap$. Let $g : X \times KY \to A$ be right iteration preserving with $g \circ (id \times \eta) = f$. To show that $g = (f \circ swap)^{\blacktriangleleft} \circ swap$, by uniqueness of $(f \circ swap)^{\blacktriangleleft}$ it suffices to show that $g \circ swap$ satisfies $g \circ swap \circ (\eta \times id) = f \circ swap$ and is left iteration preserving.

Indeed,

$$g \circ swap \circ (\eta \times id) = g \circ (id \times \eta) \circ swap = f \circ swap$$

and

$$\begin{aligned} & g \circ swap \circ (h^{\sharp} \times id) \\ = {} & g \circ (id \times h^{\sharp}) \circ swap \\ = {} & ((g + id) \circ dstl \circ (id \times h))^{\sharp} \circ swap \\ = {} & ((g \circ swap + id) \circ dstr \circ (h \times id))^{\sharp}, \qquad \textbf{(Uniformity)} \end{aligned}$$

for any $Z \in |\mathscr{C}|, h : Z \to KY + Z$.

This concludes one direction of the proof, the other direction follows symmetrically. □

Lemma 5.15. *In a Cartesian closed category every free Elgot algebra is stable.*

Proof. Let $\mathscr{C}$ be Cartesian closed and let KX be a free Elgot algebra on some $X \in |\mathscr{C}|$.

To show left stability of KX we define $f^{\blacktriangleleft} := eval \circ ((curry\ f)^{\star} \times id)$ for any $X \in |\mathscr{C}|$, $A \in |ElgotAlgs(\mathscr{C})|$, and $f : Y \times X \to A$. We will now verify that this does indeed satisfy the requisite properties, i.e.

$$\begin{aligned} & eval \circ ((curry\ f)^{\star} \times id) \circ (\eta \times id) \\ = {} & eval \circ ((curry\ f)^{\star} \circ \eta \times id) \\ = {} & eval \circ (curry\ f \times id) \\ = {} & f \end{aligned}$$

and for any $Z \in |\mathscr{C}|, h : Z \to KY + Z$:

$$\begin{aligned} & eval \circ ((curry\ f)^{\star} \times id) \circ (h^{\sharp} \times id) \\ = {} & eval \circ ((curry\ f)^{\star} \circ h^{\sharp} \times id) \\ = {} & eval \circ (curry(((eval + id) \circ dstr \circ ((((curry\ f)^{\star} + id) \circ h) \times id))^{\sharp}) \times id) \\ = {} & ((eval + id) \circ dstr \circ (((curry\ f)^{\star} + id) \circ h \times id))^{\sharp} \\ = {} & ((eval + id) \circ dstr \circ (((curry\ f)^{\star} + id) \times id) \circ (h \times id))^{\sharp} \\ = {} & ((eval + id) \circ (((curry\ f)^{\star} \times id) + id) \circ dstr \circ (h \times id))^{\sharp} \\ = {} & ((eval \circ ((curry\ f)^{\star} \times id) + id) \circ dstr \circ (h \times id))^{\sharp}. \end{aligned}$$

Lastly, we need to check uniqueness of $f^{\blacktriangleleft}$. Let us consider a left iteration preserving morphism $g : KY \times X \to A$ that satisfies $g \circ (\eta \times id) = f$. Since $curry$ is an injective mapping it suffices to show that

$$\begin{aligned} & curry\ f^{\blacktriangleleft} \\ = {} & curry(eval \circ ((curry\ f)^{\star} \times id)) \\ = {} & (curry\ f)^{\star} \\ = {} & curry\ g. \end{aligned}$$

Where the last step is the only non-trivial one. Since $(curry\ f)^{\star}$ is a unique iteration preserving morphism satisfying $(curry\ f)^{\star} \circ \eta = curry\ f$, we are left to show that g is also iteration preserving and satisfies the same property.

Indeed,

$$\begin{aligned} & curry\ g \circ h^{\sharp} \\ = {} & curry\ (g \circ (h^{\sharp} \times id)) \\ = {} & curry\ (((g + id) \circ dstr \circ (h \times id))^{\sharp}) \\ = {} & curry\ (((eval \circ (curry\ g \times id) + id) \circ dstr \circ (h \times id))^{\sharp}) \\ = {} & curry\ (((eval + id) \circ ((curry\ g \times id) + id) \circ dstr \circ (h \times id))^{\sharp}) \\ = {} & curry\ (((eval + id) \circ dstr \circ ((curry\ g + id) \times id) \circ (h \times id))^{\sharp}) \\ = {} & curry\ (((eval + id) \circ dstr \circ (((curry\ g + id) \circ h) \times id))^{\sharp}) \end{aligned}$$

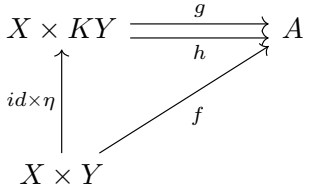
for any $Z \in |\mathscr{C}|, h : Z \to KY + Z$, and

$$\begin{aligned} & curry\ g \circ \eta \\ = {} & curry(g \circ (\eta \times id)) \\ = {} & curry\ f. \end{aligned}$$

Which completes the proof. □

For the rest of this chapter we will assume every KX to exist and be stable. Under these assumptions we show that **K** is an equational lifting monad and in fact the initial strong pre-Elgot monad. Let us first introduce a proof principle similar to the one introduced in Remark 4.9.

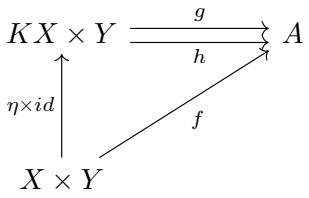
Remark 5.16 (Proof by right-stability)**.** Given two morphisms $g, h : X \times KY \to A$ where $X, Y \in |\mathscr{C}|, A \in |ElgotAlgs(\mathscr{C})|$. To show that $g = h$, it suffices to show that g and h are right iteration preserving and there exists a morphism $f : X \times Y \to A$ such that



commutes.

Of course there is also a symmetric version of this.

Remark 5.17 (Proof by left-stability)**.** Given two morphisms $g, h : KX \times Y \to A$ where $X, Y \in |\mathscr{C}|, A \in |ElgotAlgs(\mathscr{C})|$. To show that $g = h$, it suffices to show that g and h are left iteration preserving and there exists a morphism $f : X \times Y \to A$ such that

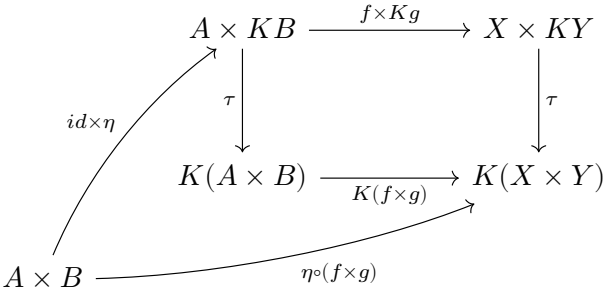


commutes.

Lemma 5.18. K *is a strong monad.*

Proof. We define strength as $\tau = (\eta : X \times Y \to K(X \times Y))^{\blacktriangleright} : X \times KY \to K(X \times Y)$. Note that by definition τ is right iteration preserving and $\tau \circ (id \times \eta) = \eta$. Most of the requisite proofs will be done by right-stability using Remark 5.16, i.e. to prove an identity we need to give a unifying morphism such that the requisite diagram commutes, and we need to show that both sides of the identity are right iteration preserving. The proofs of commutativity follow by easy rewriting and are thus deferred to the formalization. The proofs of right iteration preservation follow in most cases instantly since the morphisms are composed of (right) iteration preserving morphisms but in non-trivial cases we will give the full proof.

Naturality of τ follows by:

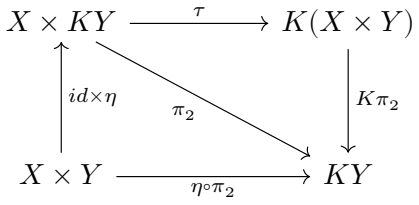


Notably, $\tau \circ (f \times Kg)$ is right iteration preserving, since for any $Z \in |\mathscr{C}|$ and $h : Z \to KY + Z$:

$$\begin{aligned}
&\tau \circ (f \times Kg) \circ (id \times h^\sharp) \\
&= \tau \circ (f \times ((Kg + id) \circ h)^\sharp) \\
&= ((\tau + id) \circ dstl \circ (id \times ((Kg + id) \circ h)))^\sharp \circ (f \times id) \\
&= (((\tau \circ (f \times Kg)) + id) \circ dstl \circ (id \times h))^\sharp. && \textbf{(Uniformity)}
\end{aligned}$$

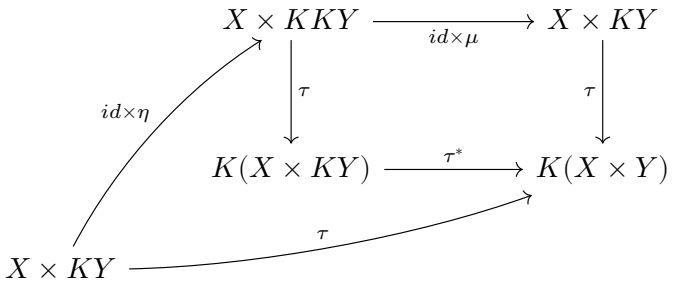
Let us now check the strength laws.

(S1) Note that for **K**, the identity $K\pi_2 \circ \tau = \pi_2$ holds more generally for any $X, Y \in |\mathscr{C}|$ instead of just for $X = 1$, which is proven by right-stability, using:

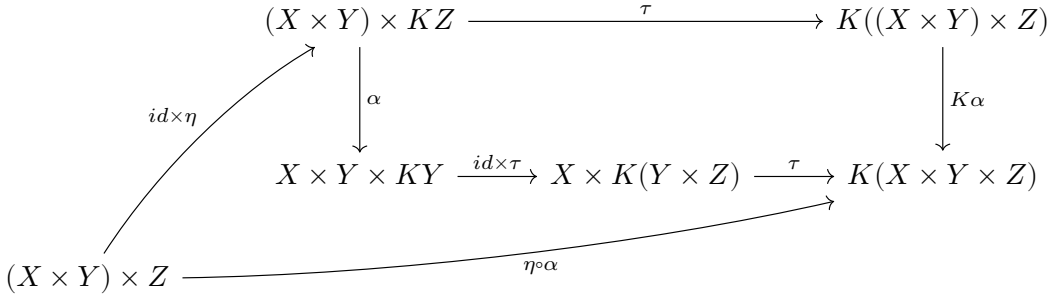


(S2) As already mentioned, $\tau \circ (id \times \eta) = \eta$ follows by definition of τ.

(S3) To show that $\tau \circ (id \times \mu) = \tau^* \circ \tau$, we will proceed by right-stability using:



(S4) Lastly, consider the following diagram for the proof by right-stability:



where $\tau \circ (id \times \tau) \circ \alpha$ is right iteration preserving, since for any $Z \in |\mathscr{C}|$ and $h : Z \to KX + Z$:

$$
\begin{aligned}
& \tau \circ (id \times \tau) \circ \alpha \circ (id \times h^\sharp) \\
&= \tau \circ \langle \pi_1 \circ \pi_1, \tau \circ \langle \pi_2 \circ \pi_1, \pi_2 \rangle \rangle \circ (id \times h^\sharp) \\
&= \tau \circ \langle \pi_1 \circ \pi_1, \tau \circ \langle \pi_2 \circ \pi_1, h^\sharp \circ \pi_2 \rangle \rangle \\
&= \tau \circ \langle \pi_1 \circ \pi_1, \tau \circ (id \times h^\sharp) \circ \langle \pi_2 \circ \pi_1, \pi_2 \rangle \rangle \\
&= \tau \circ \langle \pi_1 \circ \pi_1, ((\tau + id) \circ dstl \circ (id \times h))^\sharp \circ \langle \pi_2 \circ \pi_1, \pi_2 \rangle \rangle \\
&= \tau \circ (id \times ((\tau + id) \circ dstl \circ (id \times h))^\sharp) \circ \alpha \\
&= ((\tau + id) \circ dstl \circ (id \times ((\tau + id) \circ dstl \circ (id \times h))))^\sharp \circ \alpha \\
&= (((\tau \circ (id \times \tau) \circ \alpha) + id) \circ dstl \circ (id \times h))^\sharp. && \textbf{(Uniformity)}
\end{aligned}
$$

Thus, strength of **K** has been proven. □

As we did when proving commutativity of **D**, let us record some facts about τ and the induced σ, before proving commutativity of **K**.

Corollary 5.19. *σ is left iteration preserving and satisfies $\sigma \circ (\eta \times id) = \eta$ and the following properties of τ and σ hold.*

$$\tau \circ (f^* \times g^*) = (\tau \circ (id \times g))^* \circ \tau \circ (f^* \times id) \qquad (\tau_1)$$

$$\sigma \circ (f^* \times g^*) = (\sigma \circ (f \times id))^* \circ \sigma \circ (id \times g^*) \qquad (\sigma_1)$$

Proof. Note that the first part of the proof amounts to showing that $\sigma = \eta^\blacktriangleleft$ using uniqueness

of $\eta^{\blacktriangleleft}$. Indeed,

$$\begin{aligned}
&\sigma \circ (\eta \times id) \\
&= K swap \circ \tau \circ swap \circ (\eta \times id) \\
&= K swap \circ \tau \circ (id \times \eta) \circ swap \\
&= K swap \circ \eta \circ swap \\
&= \eta \circ swap \circ swap \\
&= \eta
\end{aligned}$$

and for any $h : Z \to KX + Z$

$$\begin{aligned}
&\sigma \circ (h^\sharp \times id) \\
&= K swap \circ \tau \circ swap \circ (h^\sharp \times id) \\
&= K swap \circ \tau \circ (id \times h^\sharp) \circ swap \\
&= K swap \circ ((\tau + id) \circ dstl \circ (id \times h))^\sharp \circ swap \\
&= (((K swap \circ \tau) + id) \circ dstl \circ (id \times h))^\sharp \circ swap \\
&= ((\sigma + id) \circ dstr \circ (h \times id))^\sharp. && \textbf{(Uniformity)}
\end{aligned}$$

Let us now proceed with the properties of τ and σ.

(τ_1)

$$\begin{aligned}
&(\tau \circ (id \times g))^* \circ \tau \circ (f^* \times id) \\
&= \tau^* \circ K(id \times g) \circ \tau \circ (f^* \times id) \\
&= \tau^* \circ \tau \circ (id \times Kg) \circ (f^* \times id) \\
&= \tau \circ (id \times \mu) \circ (id \times Kg) \circ (f^* \times id) \\
&= \tau \circ (id \times g^*) \circ (f^* \times id) \\
&= \tau \circ (f^* \times g^*)
\end{aligned}$$

(σ_1)

$$\begin{aligned}
&(\sigma \circ (f \times id))^* \circ \sigma \circ (id \times g^*) \\
&= \sigma^* \circ K(f \times id) \circ \sigma \circ (id \times g^*) \\
&= \sigma^* \circ \sigma \circ (Kf \times id) \circ (id \times g^*) \\
&= \sigma \circ (\mu \times id) \circ (Kf \times id) \circ (id \times g^*) \\
&= \sigma \circ (f^* \times id) \circ (id \times g^*) \\
&= \sigma \circ (f^* \times g^*)
\end{aligned}$$

Thus, the proof has been concluded. □

The following Lemma is central to the proof of commutativity.

Lemma 5.20. *Given $f : X \to KY + X, g : Z \to KA + Z$,*

$$\tau^* \circ \sigma \circ (((\eta + id) \circ f)^\sharp \times ((\eta + id) \circ g)^\sharp) = \sigma^* \circ \tau \circ (((\eta + id) \circ f)^\sharp \times ((\eta + id) \circ g)^\sharp).$$

Proof. Let us abbreviate $\widehat{f} := (\eta + id) \circ f$ and $\widehat{g} := (\eta + id) \circ g$. It suffices to find a

$$w : X \times Z \to K(X \times KA + KY \times Z) + X \times Z$$

such that $\widehat{f}^\sharp \circ \pi_1 = [\widehat{f}^\sharp \circ \pi_1, \eta \circ \pi_1]^* \circ w^\sharp$ and $\widehat{g}^\sharp \circ \pi_2 = [\widehat{g}^\sharp \circ \pi_2, \eta \circ \pi_2]^* \circ w^\sharp$, because then

$$
\begin{aligned}
&\tau^* \circ \sigma \circ (\widehat{f}^\sharp \times \widehat{g}^\sharp) \\
&= \tau^* \circ \sigma \circ ([\widehat{f}^\sharp \circ \pi_1, \eta \circ \pi_1]^* \times [\widehat{g}^\sharp \circ \pi_2, \eta \circ \pi_2]^*) \circ (w^\sharp \times w^\sharp) \\
&= \tau^* \circ (\sigma \circ ([\widehat{f}^\sharp \circ \pi_1, \eta \circ \pi_1] \times id))^* \circ \sigma \circ (id \times [\widehat{g}^\sharp \circ \pi_2, \eta \circ \pi_2]^*) \circ (w^\sharp \times w^\sharp) \\
&= \tau^* \circ (\sigma \circ ([\widehat{f}^\sharp \circ \pi_1, \eta \circ \pi_1] \times id))^* \circ K(id \times [\widehat{g}^\sharp \circ \pi_2, \eta \circ \pi_2]^*) \circ \sigma \circ (w^\sharp \times w^\sharp) \\
&= \tau^* \circ (\sigma \circ ([\widehat{f}^\sharp \circ \pi_1, \eta \circ \pi_1] \times id))^* \circ K(id \times [\widehat{g}^\sharp \circ \pi_2, \eta \circ \pi_2]^*) \circ Kswap \circ \tau \circ \Delta \circ w^\sharp \\
&= \tau^* \circ (\sigma \circ ([\widehat{f}^\sharp \circ \pi_1, \eta \circ \pi_1] \times id))^* \\
&\qquad \circ K(id \times [\widehat{g}^\sharp \circ \pi_2, \eta \circ \pi_2]^*) \circ Kswap \circ K\langle \eta, id \rangle \circ w^\sharp \qquad \text{(Theorem 5.22)} \\
&= \tau^* \circ (\sigma \circ ([\widehat{f}^\sharp \circ \pi_1, \eta \circ \pi_1] \times id))^* \circ K(id \times [\widehat{g}^\sharp \circ \pi_2, \eta \circ \pi_2]^*) \circ K\langle id, \eta \rangle \circ w^\sharp \\
&= \tau^* \circ (\sigma \circ ([\widehat{f}^\sharp \circ \pi_1, \eta \circ \pi_1] \times id))^* \circ K\langle id, [\widehat{g}^\sharp \circ \pi_2, \eta \circ \pi_2] \rangle \circ w^\sharp \\
&= \tau^* \circ (\sigma \circ \langle [\widehat{f}^\sharp \circ \pi_1, \eta \circ \pi_1], [\widehat{g}^\sharp \circ \pi_2, \eta \circ \pi_2] \rangle)^* \circ w^\sharp \\
&= \tau^* \circ (\sigma \circ [\langle \widehat{f}^\sharp \circ \pi_1, \eta \circ \pi_2 \rangle, \langle \eta \circ \pi_1, \widehat{g}^\sharp \circ \pi_2 \rangle])^* \circ w^\sharp \\
&= (\tau^* \circ \sigma \circ [\widehat{f}^\sharp \times \eta, \eta \times \widehat{g}^\sharp])^* \circ w^\sharp \\
&= ([\tau^* \circ \sigma \circ (\widehat{f}^\sharp \times \eta), \tau^* \circ \sigma \circ (\eta \times \widehat{g}^\sharp)])^* \circ w^\sharp \\
&= ([\tau^* \circ K(id \times \eta) \circ \sigma \circ (\widehat{f}^\sharp \times id), \tau^* \circ \eta \circ (id \times \widehat{g}^\sharp)])^* \circ w^\sharp \\
&= ([\sigma \circ (\widehat{f}^\sharp \times id), \tau \circ (id \times \widehat{g}^\sharp)])^* \circ w^\sharp,
\end{aligned}
$$

and by a symmetric argument also

$$\sigma^* \circ \tau \circ (\widehat{f}^\sharp \times \widehat{g}^\sharp) = ([\sigma \circ (\widehat{f}^\sharp \times id), \tau \circ (id \times \widehat{g}^\sharp)])^* \circ w^\sharp.$$

Note that we are referencing the equational lifting law established in Theorem 5.22 even though for a monad to be an equational lifting monad it has to be commutative first. However, since we are merely referencing the equational law, which can (and does in this case) hold without depending on commutativity, this does not pose a problem.

We are thus left to find such a w, consider

$$w \coloneqq [i_1 \circ Ki_1 \circ \tau, ((Ki_2 \circ \sigma) + id) \circ dstr \circ (\widehat{f} \times id)] \circ dstl \circ (id \times \widehat{g}).$$

w indeed satisfies the requisite properties, let us check the first property, the second one follows by a symmetric argument. We need to show that

$$[\widehat{f}^\sharp \circ \pi_1, \eta \circ \pi_1]^* \circ w^\sharp = ([i_1 \circ \pi_1, (\widehat{f}^\sharp \circ \pi_1 + id) \circ dstl] \circ dstr \circ (\widehat{f} \times g))^\sharp = \widehat{f}^\sharp \circ \pi_1.$$

Indeed,

$$
\begin{aligned}
& [\widehat{f}^{\sharp} \circ \pi_1, \eta \circ \pi_1]^{*} \circ w^{\sharp} \\
&= (([\widehat{f}^{\sharp} \circ \pi_1, \eta \circ \pi_1]^{*} + id) \circ w)^{\sharp} \\
&= ([i_1 \circ (\widehat{f}^{\sharp} \circ \pi_1)^{*} \circ \tau, ((K\pi_1 \circ \sigma) + id) \circ dstr \circ (\widehat{f} \times id)] \circ dstl \circ (id \times \widehat{g}))^{\sharp} \\
&= ([i_1 \circ (\widehat{f}^{\sharp} \circ \pi_1)^{*} \circ \tau, ((K\pi_1 \circ \sigma) + id) \circ dstr \circ (\widehat{f} \times id)] \circ dstl \circ (id \times (\eta + id)) \circ (id \times g))^{\sharp} \\
&= ([i_1 \circ (\widehat{f}^{\sharp} \circ \pi_1)^{*} \circ \tau, ((K\pi_1 \circ \sigma) + id) \circ dstr \circ (\widehat{f} \times id)] \circ ((id \times \eta) + id) \circ dstl \circ (id \times g))^{\sharp} \\
&= ([i_1 \circ (\widehat{f}^{\sharp} \circ \pi_1)^{*} \circ \tau \circ (id \times \eta), ((K\pi_1 \circ \sigma) + id) \circ dstr \circ (\widehat{f} \times id)] \circ dstl \circ (id \times g))^{\sharp} \\
&= ([i_1 \circ \widehat{f}^{\sharp} \circ \pi_1, ((K\pi_1 \circ \sigma) + id) \circ dstr \circ (\widehat{f} \times id)] \circ dstl \circ (id \times g))^{\sharp} \\
&= ([i_1 \circ [id, \widehat{f}^{\sharp}] \circ \widehat{f} \circ \pi_1, ((K\pi_1 \circ \sigma) + id) \circ dstr \circ (\widehat{f} \times id)] \circ dstl \circ (id \times g))^{\sharp} && \textbf{(Fixpoint)} \\
&= ([i_1 \circ [id, \widehat{f}^{\sharp}] \circ \pi_1 \circ (\widehat{f} \times id), ((K\pi_1 \circ \sigma) + id) \circ dstr \circ (\widehat{f} \times id)] \circ dstl \circ (id \times g))^{\sharp} \\
&= ([i_1 \circ [id, \widehat{f}^{\sharp}] \circ \pi_1, ((K\pi_1 \circ \sigma) + id) \circ dstr] \circ ((\widehat{f} \times id) + (\widehat{f} \times id)) \circ dstl \circ (id \times g))^{\sharp} \\
&= ([i_1 \circ [id, \widehat{f}^{\sharp}] \circ \pi_1, ((K\pi_1 \circ \sigma) + id) \circ dstr] \circ dstl \circ (\widehat{f} \times g))^{\sharp} \\
&= ([i_1 \circ [id, \widehat{f}^{\sharp}] \circ (\pi_1 + \pi_1) \circ dstr, ((K\pi_1 \circ \sigma) + id) \circ dstr] \circ dstl \circ (\widehat{f} \times g))^{\sharp} \\
&= ([i_1 \circ [id, \widehat{f}^{\sharp}] \circ (\pi_1 + \pi_1), ((K\pi_1 \circ \sigma) + id)] \circ (dstr + dstr) \circ dstl \circ (\widehat{f} \times g))^{\sharp} \\
&= ([i_1 \circ [id, \widehat{f}^{\sharp}] \circ (\pi_1 + \pi_1), ((K\pi_1 \circ \sigma) + id)] \circ [i_1 + i_1, i_2 + i_2] \circ (dstl + dstl) \circ dstr \circ (\widehat{f} \times g))^{\sharp} \\
&= ([[i_1 \circ [id, \widehat{f}^{\sharp}] \circ i_1 \circ \pi_1, i_1 \circ K\pi_1 \circ \sigma], [i_1 \circ [id, \widehat{f}^{\sharp}] \circ i_2 \circ \pi_1, i_2]] \circ (dstl + dstl) \circ dstr \circ (\widehat{f} \times g))^{\sharp} \\
&= ([[i_1 \circ \pi_1, i_1 \circ \pi_1], [i_1 \circ \widehat{f}^{\sharp} \circ \pi_1, i_2]] \circ (dstl + dstl) \circ dstr \circ (\widehat{f} \times g))^{\sharp} \\
&= ([i_1 \circ [\pi_1, \pi_1], (\widehat{f}^{\sharp} \circ \pi_1 + id)] \circ (dstl + dstl) \circ dstr \circ (\widehat{f} \times g))^{\sharp} \\
&= ([i_1 \circ \pi_1, (\widehat{f}^{\sharp} \circ \pi_1 + id) \circ dstl] \circ dstr \circ (\widehat{f} \times g))^{\sharp}
\end{aligned}
$$

and

$$
\begin{aligned}
& \widehat{f}^{\sharp} \circ \pi_1 \\
&= ((id + \Delta) \circ h)^{\sharp} && \textbf{(Uniformity)} \\
&= ([i_1, ((id + \Delta) \circ h)^{\sharp} + id] \circ h)^{\sharp} && \textbf{(Diamond)} \\
&= ([i_1, (\widehat{f}^{\sharp} \circ \pi_1) + id] \circ h)^{\sharp} && \textbf{(Uniformity)} \\
&= ([i_1 \circ \pi_1, ((\widehat{f} \circ \pi_1 \circ \pi_1) + (\pi_1 \times id)) \circ dstl \circ \langle id, g \circ \pi_2 \rangle] \circ dstr \circ (\widehat{f} \times id))^{\sharp} \\
&= ([i_1 \circ \pi_1, ((\widehat{f} \circ \pi_1) + id) \circ ((\pi_1 \times id) + (\pi_1 \times id)) \circ dstl \circ \langle id, g \circ \pi_2 \rangle] \circ dstr \circ (\widehat{f} \times id))^{\sharp} \\
&= ([i_1 \circ \pi_1, ((\widehat{f} \circ \pi_1) + id) \circ dstl \circ \langle \pi_1, g \circ \pi_2 \rangle] \circ dstr \circ (\widehat{f} \times id))^{\sharp} \\
&= ([i_1 \circ \pi_1 \circ (id \times g), ((\widehat{f} \circ \pi_1) + id) \circ dstl \circ (id \times g)] \circ dstr \circ (\widehat{f} \times id))^{\sharp} \\
&= ([i_1 \circ \pi_1, ((\widehat{f} \circ \pi_1) + id) \circ dstl] \circ dstr \circ (\widehat{f} \times g))^{\sharp},
\end{aligned}
$$

where $h = (\pi_1 + (\pi_1 + (\pi_1 \times id)) \circ dstl \circ \langle id, g \circ \pi_2 \rangle) \circ dstr \circ (\widehat{f} \times id)$ and the application of

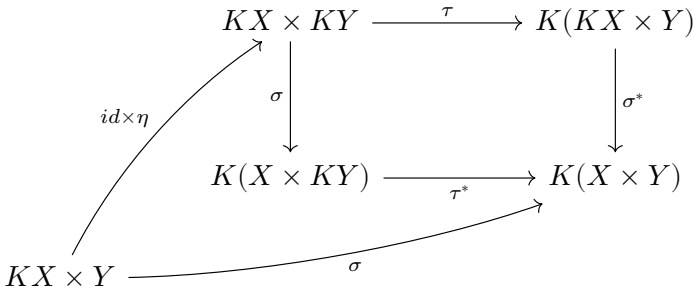
Uniformity is justified, since

$$\begin{aligned}
&(id + \pi_1) \circ (id + \Delta) \circ h \\
&= (\pi_1 + ((\pi_1 \circ [\pi_1, \pi_1 \times id]) \circ dstl \circ \langle id, g \circ \pi_2 \rangle)) \circ dstr \circ (\hat{f} \times id) \\
&= (\pi_1 + ([\pi_1 \circ \pi_1, \pi_1 \circ \pi_1] \circ dstl \circ \langle id, g \circ \pi_2 \rangle)) \circ dstr \circ (\hat{f} \times id) \\
&= (\pi_1 + (\pi_1 \circ \pi_1 \circ \langle id, g \circ \pi_2 \rangle)) \circ dstr \circ (\hat{f} \times id) \\
&= (\pi_1 + \pi_1) \circ dstr \circ (\hat{f} \times id) \\
&= \pi_1 \circ (\hat{f} \times id) \\
&= \hat{f} \circ \pi_1.
\end{aligned}$$

This concludes the proof. □

Lemma 5.21. K *is a commutative monad.*

Proof. We need to show that $\tau^* \circ \sigma = \sigma^* \circ \tau : KX \times KY \to K(X \times Y)$. Let us proceed by right stability, consider the following diagram.



The diagram commutes since

$$\sigma^* \circ \tau \circ (id \times \eta) = \sigma^* \circ \eta = \sigma$$

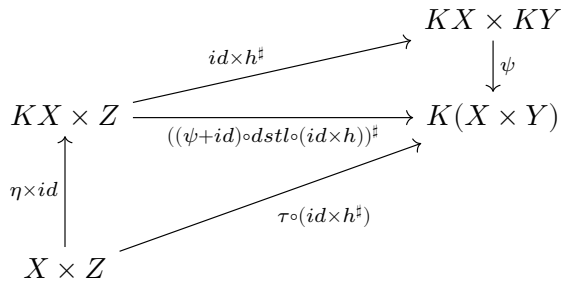
and

$$\tau^* \circ \sigma \circ (id \times \eta) = \tau^* \circ K(id \times \eta) \circ \sigma = (\tau \circ (id \times \eta))^* \circ \sigma = \sigma.$$

We are left to show that both $\sigma^* \circ \tau$ and $\tau^* \circ \sigma$ are right iteration preserving. Let $h : Z \to KY + Z$, indeed

$$\sigma^* \circ \tau \circ (id \times h^\sharp) = \sigma^*((\tau + id) \circ dstl \circ (id \times h))^\sharp = (((\sigma^* \circ \tau) + id) \circ dstl \circ (id \times h))^\sharp.$$

Let $\psi := \tau^* \circ \sigma$ and let us proceed by left stability to show that ψ is right iteration preserving, consider the following diagram



which commutes, since

$$\begin{aligned}
&\psi \circ (id \times h^\sharp) \circ (\eta \times id) \\
&= \psi \circ (\eta \times id) \circ (id \times h^\sharp) \\
&= \tau^* \circ \eta \circ (id \times h^\sharp) \\
&= \tau \circ (id \times h^\sharp) \\
&= ((\tau + id) \circ dstl \circ (id \times h))^\sharp \\
&= ((\psi + id) \circ dstl \circ (id \times h))^\sharp \circ (\eta \times id). && (\textbf{Uniformity})
\end{aligned}$$

We are left to show that both $\psi \circ (id \times h^\sharp)$ and $((\psi + id) \circ dstl \circ (id \times h))^\sharp$ are left iteration preserving. Let $g : A \to KX + A$, then $\psi \circ (id \times h^\sharp)$ is left iteration preserving, since

$$\begin{aligned}
&\psi \circ (id \times h^\sharp) \circ (g^\sharp \times id) \\
&= \psi \circ (g^\sharp \times id) \circ (id \times h^\sharp) \\
&= \tau^* \circ ((\sigma + id) \circ dstr \circ (g \times id))^\sharp \circ (id \times h^\sharp) \\
&= ((\psi + id) \circ dstr \circ (g \times id))^\sharp \circ (id \times h^\sharp) \\
&= (((\psi \circ (id \times h^\sharp)) + id) \circ dstr \circ (g \times id))^\sharp. && (\textbf{Uniformity})
\end{aligned}$$

Lastly, we need to show that

$$((\psi + id) \circ dstl \circ (id \times h))^\sharp \circ (g^\sharp \times id) = ((((\psi + id) \circ dstl \circ (id \times h))^\sharp + id) \circ dstr \circ (g \times id))^\sharp.$$

Note that by **Uniformity** the left-hand side can be rewritten as

$$((((\psi + id) \circ dstr \circ (g \times id))^\sharp + id) \circ dstl \circ (id \times h))^\sharp.$$

Consider now, that

$$\begin{aligned}
&((((\psi + id) \circ dstl \circ (id \times h))^\sharp + id) \circ dstr \circ (g \times id))^\sharp \\
&= (((((\psi + id) \circ dstl \circ (id \times h))^\sharp)^* + id) \circ (\eta + id) \circ dstr \circ (g \times id))^\sharp \\
&= (((\psi + id) \circ dstl \circ (id \times h))^\sharp)^* \circ ((\eta + id) \circ dstr \circ (g \times id))^\sharp \\
&= (((\psi + id) \circ dstl \circ (id \times h))^\sharp)^* \circ (((\sigma \circ (\eta \times id)) + id) \circ dstr \circ (g \times id))^\sharp \\
&= (((\psi + id) \circ dstl \circ (id \times h))^\sharp)^* \circ \sigma \circ (((\eta + id) \circ g)^\sharp \times id) \\
&= (((\psi^* + id) \circ (\eta + id) \circ dstl \circ (id \times h))^\sharp)^* \circ \sigma \circ (((\eta + id) \circ g)^\sharp \times id) \\
&= \psi^* \circ (((\eta + id) \circ dstl \circ (id \times h))^\sharp)^* \circ \sigma \circ (((\eta + id) \circ g)^\sharp \times id) \\
&= \psi^* \circ (((\eta + id) \circ dstl \circ (id \times h))^\sharp)^* \circ \sigma \circ (((\eta + id) \circ g)^\sharp \times id) \\
&= \psi^* \circ ((((\tau \circ (id \times \eta)) + id) \circ dstl \circ (id \times h))^\sharp)^* \circ \sigma \circ (((\eta + id) \circ g)^\sharp \times id) \\
&= \psi^* \circ (((\tau + id) \circ dstl \circ (id \times ((\eta + id) \circ h)))^\sharp)^* \circ \sigma \circ (((\eta + id) \circ g)^\sharp \times id) \\
&= \psi^* \circ (\tau \circ (id \times ((\eta + id) \circ h)^\sharp))^* \circ \sigma \circ (((\eta + id) \circ g)^\sharp \times id) \\
&= \psi^* \circ \tau^* \circ K(id \times ((\eta + id) \circ h)^\sharp) \circ \sigma \circ (((\eta + id) \circ g)^\sharp \times id) \\
&= \psi^* \circ \tau^* \circ \sigma \circ (id \times ((\eta + id) \circ h)^\sharp) \circ (((\eta + id) \circ g)^\sharp \times id) \\
&= \psi^* \circ \tau^* \circ \sigma \circ (((\eta + id) \circ g)^\sharp \times ((\eta + id) \circ h)^\sharp),
\end{aligned}$$

and by a symmetric argument

$$\begin{aligned}&((((\psi + id) \circ dstr \circ (g \times id))^\sharp + id) \circ dstl \circ (id \times h))^\sharp \\ &= \psi^* \circ \sigma^* \circ \tau \circ (((\eta + id) \circ g)^\sharp \times ((\eta + id) \circ h)^\sharp).\end{aligned}$$

We are thus done by

$$\begin{aligned}&((\psi + id) \circ dstl \circ (id \times h))^\sharp \circ (g^\sharp \times id) \\ &= ((((\psi + id) \circ dstr \circ (g \times id))^\sharp + id) \circ dstl \circ (id \times h))^\sharp && \textbf{(Uniformity)} \\ &= \psi^* \circ \sigma^* \circ \tau \circ (((\eta + id) \circ g)^\sharp \times ((\eta + id) \circ h)^\sharp) \\ &= \psi^* \circ \tau^* \circ \sigma \circ (((\eta + id) \circ g)^\sharp \times ((\eta + id) \circ h)^\sharp) && \text{(Lemma 5.20)} \\ &= ((((\psi + id) \circ dstl \circ (id \times h))^\sharp + id) \circ dstr \circ (g \times id))^\sharp. && \square\end{aligned}$$

Theorem 5.22. K *is an equational lifting monad.*

Proof. Since we have already shown commutativity, we are left to show that $\tau \circ \Delta = K\langle \eta, id\rangle$. Note that $K\langle \eta, id\rangle = (\eta \circ \langle \eta, id\rangle)^\star$, which is the unique Elgot algebra morphism satisfying $K\langle \eta, id\rangle \circ \eta = \eta \circ \langle \eta, id\rangle$. It thus suffices to show that $\tau \circ \Delta$ satisfies the same identity and is iteration preserving.

The identity follows easily:

$$\begin{aligned}&\tau \circ \Delta \circ \eta \\ &= \tau \circ \langle \eta, \eta\rangle \\ &= \tau \circ (id \times \eta) \circ \langle \eta, id\rangle \\ &= \eta \circ \langle \eta, id\rangle.\end{aligned}$$

For iteration preservation of $\tau \circ \Delta$ consider $Z \in |\mathscr{C}|$ and $h : Z \to KX + Z$, then

$$\begin{aligned}&\tau \circ \Delta \circ h^\sharp \\ &= \tau \circ \langle h^\sharp, h^\sharp\rangle \\ &= \tau \circ (id \times h^\sharp) \circ \langle h^\sharp, id\rangle \\ &= ((\tau + id) \circ dstl \circ (id \times f))^\sharp \circ \langle h^\sharp, id\rangle \\ &= (((\tau \circ \Delta) + id) \circ f)^\sharp. && \textbf{(Uniformity)}\end{aligned}$$

Note that by monicity of $dstl^{-1}$ and by **Fixpoint**

$$(\Delta + \langle f^\sharp, id\rangle) \circ f = dstl \circ \langle f^\sharp, f\rangle. \qquad (*)$$

The application of **Uniformity** is then justified by

$$\begin{aligned}&(id + \langle f^\sharp, id\rangle) \circ ((\tau \circ \Delta) + id) \circ f \\ &= ((\tau \circ \Delta) + \langle f^\sharp, id\rangle) \circ f \\ &= (\tau + id) \circ (\Delta + \langle f^\sharp, id\rangle) \circ f \\ &= (\tau + id) \circ dstl \circ \langle f^\sharp, f\rangle && (*) \\ &= (\tau + id) \circ dstl \circ (id \times f) \circ \langle f^\sharp, id\rangle. && \square\end{aligned}$$

Theorem 5.23. **K** *is the initial (strong) pre-Elgot monad.*

Proof. Note that **K** is a pre-Elgot monad by definition and strong pre-Elgot by Lemma 5.18. Let us first show that **K** is the initial pre-Elgot monad.

Given any pre-Elgot monad **T**, let us introduce alternative names for the monad operations of **T** and **K** to avoid confusion:

$$\mathbf{T} = (T, \eta^T, \mu^T)$$

and

$$\mathbf{K} = (K, \eta^K, \mu^T).$$

For every $X \in |\mathscr{C}|$ we define $¡ = (\eta^T : X \to TX)^\star : KX \to TX$. Note that ¡ is per definition the unique iteration preserving morphism that satisfies $¡ \circ \eta^K = \eta^T$. We are done after showing that ¡ is natural and respects the monad multiplication.

Let $f : X \to Y$. For naturality of ¡ it suffices to show

$$¡ \circ Kf = (Tf \circ \eta^T)^\star = Tf \circ ¡,$$

where $(Tf \circ \eta^T)^\star$ is the unique Elgot algebra morphism satisfying $(Tf \circ \eta^T)^\star \circ \eta^K = Tf \circ \eta^T$. Note that both $¡ \circ Kf$ and $Tf \circ ¡$ are iteration preserving since they are composed of iteration preserving morphisms and both satisfy the requisite property, since $Tf \circ ¡ \circ \eta^K = Tf \circ \eta^T$ follows instantly and

$$\begin{aligned} & ¡ \circ Kf \circ \eta^K \\ = {} & ¡ \circ \eta^K \circ f \\ = {} & \eta^T \circ f \\ = {} & Tf \circ \eta^T. \end{aligned}$$

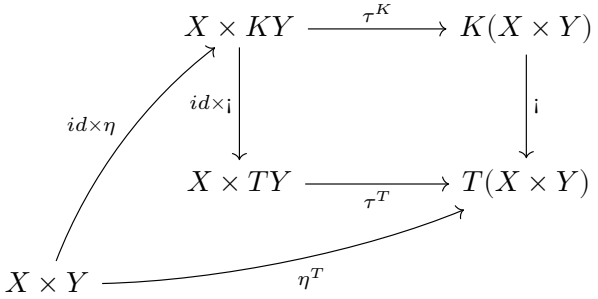
Let us proceed similarly for showing that ¡ respects the monad multiplication, i.e. consider

$$¡ \circ \mu = ¡^\star = \mu^T \circ T¡ \circ ¡,$$

where $¡^\star$ is the unique Elgot algebra morphism satisfying $¡^\star \circ \eta^K = ¡$. Note that again both sides of the identity are iteration preserving, since they are composed of iteration preserving morphisms. Consider also that $¡ \circ \mu^K \circ \eta^K = ¡$ and

$$\begin{aligned} & \mu^T \circ T¡ \circ ¡ \circ \eta^K \\ = {} & \mu^T \circ ¡ \circ K¡ \circ \eta^K \\ = {} & \mu^T \circ ¡ \circ \eta^K \circ ¡ \\ = {} & \mu^T \circ \eta^T \circ ¡ \\ = {} & ¡. \end{aligned}$$

Thus, **K** is an initial pre-Elgot monad. To show that **K** is also initial strong pre-Elgot, assume that **T** is strong with strength τ^T and let us call the strength of **K** τ^K. We are left to show that ¡ respects strength, i.e. $¡ \circ \tau^K = \tau^T \circ (id \times ¡) : X \times KY \to T(X \times Y)$. We proceed by right-stability, using:



The diagram commutes, since ¡ $\circ \tau^K = \eta^T = \tau^T \circ (id \times \eta^T) = \tau^T \circ (id \times$ ¡$) \circ (id \times \eta^T)$. Now we are done, since ¡ $\circ \tau^K$ and $\tau^T \circ (id \times$ ¡$)$ are both right iteration preserving because both are composed of (right) iteration preserving morphisms. □

6 A Case Study on Setoids

In Chapter 4 we have argued that the delay monad is not an equational lifting monad, because it does not only model partiality, but it also considers computation time in its built-in notion of equality. One way to remedy this is to take the quotient of the delay monad where computations with the same result are identified. In this chapter we will use the quotients-as-setoid approach, i.e. we will work in the category of setoids and show that the quotiented delay monad is an instance of the previously defined monad **K** in this category.

6.1 Setoids in Type Theory

We will now introduce the category that the rest of the chapter will take place in. Let us start with some basic definitions.

Definition 6.1 (Setoid)**.** A setoid is a tuple $(A, \overset{A}{=})$ where A (usually called the *carrier*) is a type and $\overset{A}{=}$ is an equivalence relation on the inhabitants of A.

For brevity, we will not use the tuple notation most of the time, instead we will just say 'Let A be a setoid' and implicitly call the equivalence relation $\overset{A}{=}$.

Definition 6.2 (Setoid Morphism)**.** A morphism between setoids A and B constitutes a function $f : A \to B$ between the carriers, such that f respects the equivalences, i.e. for any $x, y : A$, $x \overset{A}{=} y$ implies $f\ x \overset{B}{=} f\ y$. We will denote setoid morphisms as $A \rightsquigarrow B$.

Let us now consider the function space setoid, which is of special interest, since it carries a notion of equality between functions.

Definition 6.3 (Function Space Setoid)**.** Given two setoids A and B, the function space setoid on these setoids is defined as $(A \rightsquigarrow B, \doteq)$ or just $A \rightsquigarrow B$, where $\doteq$ is the point wise equality on setoid morphisms.

Setoids together with setoid morphisms form a category that we will call *Setoids*. Properties of *Setoids* have already been examined in [15], however we will reiterate some of these properties now to introduce notation that will be used for the rest of the chapter.

Proposition 6.4. *Setoids is a distributive category.*

Proof. To show that *Setoids* is (co)Cartesian we will give the respective data types and unique functions. For brevity, we will omit the proofs that the functions respect the corresponding equivalences, these are however included in the Agda standard library [21].

- **Products**:

```
record _×_ {a b} (A : Set a) (B : Set b) : Set (a ⊔ b) where
  constructor _,_
  field
    fst : A
    snd : B

<_,_> : ∀ {a b c} {A : Set a} {B : Set b} {C : Set c}
        → (A → B) → (A → C) → A → (B × C)
< f , g > x = (f x , g x)
```

The product setoid is denoted $(A \times B, \overset{\times}{=})$ or just $A \times B$. Equality of products is defined in the canonical way.

- **Terminal Object**:

```
record ⊤ {l} : Set l where
  constructor tt

! : ∀ {l} {X : Set l} → X → ⊤ {l}
! _ = tt
```

The terminal setoid is thus $(\top, \overset{\top}{=})$, where $\top \overset{\top}{=} \top$.

- **Coproducts**:

```
data _+_ {a b} (A : Set a) (B : Set b) : Set (a ⊔ b) where
  i₁ : A → A + B
  i₂ : B → A + B

[_,_] : ∀ {a b c} {A : Set a} {B : Set b} {C : Set c}
        → (A → C) → (B → C) → (A + B) → C
[ f , g ] (i₁ x) = f x
[ f , g ] (i₂ x) = g x
```

Similarly to products, the coproduct setoid is denoted $(A + B, \overset{+}{=})$ or just $A + B$, where equality of coproducts is defined in the canonical way.

- **Initial Object**:

```
data ⊥ {l} : Set l where

¡ : ∀ {l} {X : Set l} → ⊥ {l} → X
¡ ()
```

The initial setoid is then $(\bot, \emptyset)$, where the equivalence is the empty relation.

Lastly we need to show that the canonical distributivity function is an iso. Recall that the canonical distributivity morphism is defined as $dstl^{-1} = [id \times i_1, id \times i_2] : A \times B + A \times C \to A \times (B + C)$. This is equivalent to the following definition that uses pattern matching.

```
distribute¹⁻¹ : ∀ {a b c} {A : Set a} {B : Set b} {C : Set c}
                → (A × B) + (A × C) → A × (B + C)
distribute¹⁻¹ (i₁ (x , y)) = (x , i₁ y)
distribute¹⁻¹ (i₂ (x , y)) = (x , i₂ y)
```

The inverse can then be defined similarly:

```
distribute¹ : ∀ {a b c} {A : Set a} {B : Set b} {C : Set c}
              → A × (B + C) → (A × B) + (A × C)
distribute¹ (x , i₁ y) = i₁ (x , y)
distribute¹ (x , i₂ y) = i₂ (x , y)
```

Note that these functions are inverse by definition, and it follows quickly that they are setoid morphisms. □

Proposition 6.5. *Setoids is Cartesian closed.*

Proof. Let A and B be two setoids. The function space setoid $A \rightsquigarrow B$ is an exponential object of A and B, together with the functions *curry* and *eval* defined in the following listing.

```
curry : ∀ {a b c} {A : Set a} {B : Set b} {C : Set c}
        → (C × A → B) → C → A → B
curry f x y = f (x , y)

eval : ∀ {a b} {A : Set a} {B : Set b} → ((A → B) × A) → B
eval (f , x) = f x
```

The universal property of exponential objects follows instantly. □

6.2 Quotienting the Delay Monad

In this section we will introduce data types only using inference rules. For that we adopt the convention that coinductive types are introduced by doubled lines while inductive types are introduced with a single line.

Now, recall from previous chapters that Capretta's delay monad [12] is a coinductive type defined by the two constructors:

$$\frac{x : A}{now\ x : D\ A} \qquad \frac{x : D\ A}{later\ x : D\ A}$$

Furthermore, let us recall two different notions of bisimilarity between inhabitants of the delay type that have been studied previously in [16]. Afterwards, we will reiterate some facts that have been proven in [16] to then finally prove that the quotiented delay type extends to an instance of the monad **K** that has been introduced in Chapter 5.

Let A be a setoid. Lifting the equivalence $\overset{A}{=}$ to $D\ A$ yields another equivalence called *strong bisimilarity*. This equivalence is defined by the rules

$$\frac{x \overset{A}{=} y}{x \sim y} \qquad \frac{x \sim y}{later\ x \sim later\ y}$$

Proposition 6.6 ([16])**.** $(D\ A, \sim)$ *is a setoid and admits a monad structure.*

Computations in $(D\ A, \sim)$ are only identified if they evaluate to the same result in the same number of steps. In many contexts this behavior is too intensional. Instead, we will now consider the quotient of this setoid, where all computations that evaluate to the same result are identified. Let us first define a relation that states that two computations evaluate to the same result

$$\frac{x \overset{A}{=} y}{now\ x \downarrow y} \qquad \frac{x \downarrow c}{later\ x \downarrow c}\ .$$

Now, we call two computations p and q *weakly bisimilar* or $p \approx q$ if they evaluate to the same result, or don't evaluate at all, which is specified by the rules

$$\frac{a \overset{A}{=} b \qquad x \downarrow a \qquad y \downarrow b}{x \approx y} \qquad \frac{x \approx y}{later\ x \approx later\ y}$$

Proposition 6.7 ([12])**.** $(D\ A, \approx)$ *is a setoid and admits a monad structure.*

Proof. The monad unit is the constructor $now : A \to D\ A$ and the multiplication $\mu : D\ D\ A \to D\ A$ can be defined as follows:

$$\mu\ x = \begin{cases} z & \text{if } x = now\ z \\ later(\mu\ z) & \text{if } x = later\ z \end{cases}$$

Given a function $f : A \to B$, the lifted function $Df : D\ A \to D\ B$ is defined as

$$Df\ x = \begin{cases} now(f\ z) & \text{if } x = now\ z \\ later(Df\ z) & \text{if } x = later\ z \end{cases}$$

It has been shown in [12] that this indeed extends to a monad. □

For the rest of this chapter we will abbreviate $\tilde{D}\ A = (D_A, \sim)$ and $\tilde{\tilde{D}}\ A = (D_A, \approx)$.

Lemma 6.8. *Every $\tilde{\tilde{D}}\ A$ can be equipped with an Elgot algebra structure.*

Proof. We need to show that for every setoid A the resulting setoid $\tilde{\tilde{D}}\ A$ extends to an Elgot algebra.

Let X be a setoid and $f : X \nrightarrow \tilde{\tilde{D}}\ A + X$ be a setoid morphism, we define $f^\sharp : X \nrightarrow \tilde{\tilde{D}}\ A$ point wise:

$$f^\sharp\ x := \begin{cases} a & \text{if } f\ x = i_1(a) \\ later\ (f^\sharp\ a) & \text{if } f\ x = i_2(a) \end{cases}$$

Let us first verify that $f^\sharp$ is indeed a setoid morphism, i.e. given $x, y : X$ with $x \overset{X}{=} y$, we need to show that $f^\sharp\ x \approx f^\sharp\ y$. Since f is a setoid morphism we know that $f\ x \overset{+}{=} f\ y$, which already implies that $f^\sharp\ x \approx f^\sharp\ y$ by the definition of $f^\sharp$. Note that by the same argument we can define an iteration operator that respects strong bisimilarity, let us call it $f^{\tilde{\sharp}}$ as we will later need to distinguish between $f^\sharp$ and $f^{\tilde{\sharp}}$.

Next, we check the iteration laws:

- **Fixpoint**: We need to show that $f^\sharp\ x \approx [id, f^\sharp](f\ x)$ for any $x : X$. Let us proceed by case distinction:

  Case 1. $f\ x = i_1\ a$

  $$f^\sharp\ x \approx a \approx [id, f^\sharp](i_1\ a) \approx [id, f^\sharp](f\ x)$$

  Case 2. $f\ x = i_2\ a$

  $$f^\sharp\ x \approx later(f^\sharp\ a) \approx f^\sharp\ a \approx [id, f^\sharp](i_2\ a) \approx [id, f^\sharp](f\ x)$$

- **Uniformity**: Let Y be a setoid and $g : Y \rightsquigarrow \tilde{\tilde{D}}\ A + Y, h : X \rightsquigarrow Y$ be setoid morphisms, such that $(id + h) \circ f \doteq g \circ h$. We need to show that $f^\sharp\ x \approx g^\sharp(h\ x)$, for any $x : X$. Let us proceed by case distinction over $f\ x$ and $g(h\ x)$, note that by the requisite equation $(id + h) \circ f \doteq g \circ h$, we only need to consider two cases:

  Case 1. $f\ x = i_1\ a$ and $g(h\ x) = i_1\ b$
  Consider that $(id + h) \circ f \doteq g \circ h$ on x yields $i_1\ a \overset{+}{=} i_1\ b$ and thus $a \approx b$. Then indeed,

  $$f^\sharp\ x \approx a \approx b \approx g^\sharp(h\ x)$$

  Case 2. $f\ x = i_2\ a$ and $g(h\ x) = i_2\ b$
  Note that $(id + h) \circ f \doteq g \circ h$ on x yields $i_2(h\ a) \overset{+}{=} i_2\ b$ and thus $h\ a \overset{Y}{=} b$. We are done by coinduction, which yields

  $$f^\sharp\ x \approx later(f^\sharp\ a) \approx later(g^\sharp(h\ a)) \approx later(g^\sharp\ b) \approx g^\sharp(h\ x).$$

- **Folding**: Let Y be a setoid and $h : Y \rightsquigarrow X + Y$ a setoid morphism, we need to show that $(f^\sharp + h)^\sharp\ z \approx [(id + i_1) \circ f, i_2 \circ h]^\sharp\ z$ for any $z : X + Y$. Let us first establish the following fact

  $$f^\sharp\ c \approx [(id + i_1) \circ f, i_2 \circ h]^\sharp(i_1\ c) \qquad \text{for any } c : X, \tag{*}$$

  which follows by case distinction on $f\ c$ and coinduction:

  Case 1. $f\ c = i_1\ a$

  $$f^\sharp\ c \approx a \approx [(id + i_1) \circ f, i_2 \circ h]^\sharp(i_1\ c)$$

  Case 2. $f\ c = i_2\ a$

  $$f^\sharp\ c \approx later(f^\sharp\ a) \approx later([(id + i_1) \circ f, i_2 \circ h]^\sharp(i_1\ a)) \approx [(id + i_1) \circ f, i_2 \circ h]^\sharp(i_1\ c)$$

  We will now proceed with the proof of **Folding**, by case distinction on z:

  Case 1. $z = i_1\ x$
  Another case distinction on $f\ x$ yields:

  Subcase (i): $f\ x = i_1\ a$
  We are done, since $(f^\sharp + h)^\sharp(i_1\ x) \approx a \approx [(id + i_1) \circ f, i_2 \circ h]^\sharp(i_1\ x)$

  Subcase (ii): $f\ x = i_2\ a$
  Now, using the fact we established prior

  $$\begin{aligned} & (f^\sharp + h)^\sharp(i_1\ x) \\ \approx\ & later(f^\sharp\ a) \\ \approx\ & later([(id + i_1) \circ f, i_2 \circ h]^\sharp(i_1\ a)) \qquad (*) \\ \approx\ & [(id + i_1) \circ f, i_2 \circ h]^\sharp(i_1\ x). \end{aligned}$$

Case 2. $z = i_2\ y$
Let us proceed by discriminating on $h\ y$.

Subcase (i): $h\ y = i_1\ a$
Indeed by coinduction,

$$\begin{aligned}
&(f^\sharp + h)^\sharp(i_2\ y)\\
\approx\ &later((f^\sharp + h)(i_1\ a))\\
\approx\ &later([(id + i_1) \circ f, i_2 \circ h]^\sharp(i_1\ a))\\
\approx\ &[(id + i_1) \circ f, i_2 \circ h]^\sharp(i_2\ y)
\end{aligned}$$

Subcase (ii): $h\ y = i_2\ a$
Similarly by coinduction,

$$\begin{aligned}
&(f^\sharp + h)^\sharp(i_2\ y)\\
\approx\ &later((f^\sharp + h)(i_2\ a))\\
\approx\ &later([(id + i_1) \circ f, i_2 \circ h]^\sharp(i_2\ a))\\
\approx\ &[(id + i_1) \circ f, i_2 \circ h]^\sharp(i_2\ y)
\end{aligned}$$

This concludes the proof that every $\tilde{\tilde{D}}\ A$ extends to an Elgot algebra. □

In the next proof a notion of *discretized* setoid is needed, i.e. given a setoid Z, we can discretize Z by replacing the equivalence relation with propositional equality, yielding $|Z| := (Z, \equiv)$. Now, the following corollary describes how to transform an iteration on $\tilde{\tilde{D}}\ A$ into an iteration on $\tilde{D}\ A$.

Corollary 6.9. *Given a setoid morphism $g : X \rightsquigarrow \tilde{\tilde{D}}\ A + X$, there exists a setoid morphism $\bar{g} : |X| \rightsquigarrow \tilde{D}\ A + |X|$ such that $g^\sharp\ x \sim \bar{g}^{\tilde{\sharp}}\ x$ for any $x : X$.*

Proof. It is clear that propositional equality implies strong bisimilarity and thus $\bar{g}$ is a setoid morphism that behaves as g does but with a different type profile. The requisite property follows by case distinction on $g\ x$.

Case 1. $g\ x = i_1\ a$
We are done, since $g^\sharp\ x \sim a \sim \bar{g}^{\tilde{\sharp}}\ x$

Case 2. $g\ x = i_2\ a$
By coinduction $g^\sharp\ x \sim later(g^\sharp\ a) \sim later(\bar{g}^{\tilde{\sharp}}\ a) \sim \bar{g}^{\tilde{\sharp}}\ x$, which concludes the proof. □

Theorem 6.10. *Every $\tilde{\tilde{D}}\ A$ can be equipped with a free Elgot algebra structure.*

Proof. We build on Lemma 6.8, it thus suffices to show that for any setoid A, the Elgot algebra $(\tilde{\tilde{D}}\ A, (-)^{\tilde{\sharp}})$ together with the setoid morphism $now : A \rightsquigarrow \tilde{\tilde{D}}\ A$ is a free such algebra. Given an Elgot algebra $(B, (-)^{\sharp_b})$ and a setoid morphism $f : A \rightsquigarrow B$. We need to define an Elgot algebra morphism $f^\star : \tilde{\tilde{D}}\ A \rightsquigarrow B$. Consider $g : \tilde{D}\ A \rightsquigarrow B + \tilde{D}\ A$ defined by

$$g\ x = \begin{cases} i_1(f\ a) & \text{if } x = now\ a \\ i_2\ a & \text{if } x = later\ a \end{cases}$$

g trivially respects strong bisimilarity, thus consider $g^{\sharp_b} : \tilde{D}\ A \nrightarrow B$. We need to show that $g^{\sharp_b}$ also respects weak bisimilarity, thus yielding the requisite function $f^\star = g^{\sharp_b} : \tilde{\tilde{D}}\ A \nrightarrow B$. However, the proof turns out to be rather complex, let us postpone it to Corollary 6.13.

Instead, we will continue with the proof. Let us now show that $g^{\sharp_b}$ is iteration preserving. Given a setoid morphism $h : X \nrightarrow \tilde{\tilde{D}}\ A + X$, we need to show that $g^{\sharp_b}(h^\sharp\ x) \overset{B}{=} ((g^{\sharp_b} + id) \circ h)^{\sharp_b}\ x$ for any $x : X$. Using Corollary 6.9 we will proceed to show

$$g^{\sharp_b}(h^\sharp\ x) \overset{B}{=} ((g^{\sharp_b} + id) \circ \bar{h})^{\sharp_b}\ x \overset{B}{=} ((g^{\sharp_b} + id) \circ h)^{\sharp_b}\ x.$$

The second step instantly follows by **Uniformity**, considering that the identity function easily extends to a setoid morphism $id : |X| \nrightarrow X$, and thus the second step can be reduced to $((g^{\sharp_b} + id) \circ \bar{h})^{\sharp_b}\ x \overset{B}{=} ((g^{\sharp_b} + id) \circ h)^{\sharp_b}(id\ x)$. For the first step consider

$$\begin{aligned}
& g^{\sharp_b}(h^\sharp\ x) \\
\overset{B}{=}\ & g^{\sharp_b}(\bar{h}^{\tilde{\sharp}}\ x) && \text{(Corollary 6.9)} \\
\overset{B}{=}\ & (g^{\sharp_b} \circ [id, \bar{h}^{\tilde{\sharp}}])(i_2\ x) \\
\overset{B}{=}\ & ([(id + i_1) \circ g, i_2 \circ i_2] \circ [i_1, h])^{\sharp_b}(i_2\ x) && \textbf{(Uniformity)} \\
\overset{B}{=}\ & ((g^{\sharp_b} + id) \circ h)^{\sharp_b}\ x. && \textbf{(Compositionality)}
\end{aligned}$$

Thus, $g^{\sharp_b}$ is an Elgot algebra morphism. We are left to check that $g^{\sharp_b}$ satisfies the requisite properties of free objects. First, note that $g^{\sharp_b} \circ now \doteq [id, g_b^\sharp] \circ g \circ now \doteq f$ by **Fixpoint** and the definition of g. Next, we need to check uniqueness of $g^{\sharp_b}$. It suffices to show that any two Elgot algebra morphisms $e, h : \tilde{\tilde{D}}\ A \nrightarrow B$ satisfying $e \circ now \doteq f$ and $h \circ now \doteq f$ are equal.

First, note that the identity function extends to the following conversion setoid morphism $conv : \tilde{D}\ A \nrightarrow \tilde{\tilde{D}}\ A$, since strong bisimilarity implies weak bisimilarity. Furthermore, consider the setoid morphism $o : \tilde{D}\ A \nrightarrow \tilde{D}\ A + \tilde{D}\ A$ defined by

$$o\ x := \begin{cases} i_1(now\ z) & \text{if } x = now\ z \\ i_2\ z & \text{if } x = later\ z \end{cases}$$

Now, by coinduction we can easily follow that

$$x \approx ((conv + id) \circ o)^\sharp\ x \qquad \text{for any } x : D\ A. \tag{$*$}$$

Let us now return to the proof of uniqueness. We proceed by

$$\begin{aligned}
& e\ x \\
\approx\ & e(((conv + id) \circ o)^\sharp\ x) && (*) \\
\approx\ & ((e \circ conv + id) \circ o)^{\sharp_b}\ x && \text{(Preservation)} \\
\approx\ & ((h \circ conv + id) \circ o)^{\sharp_b}\ x \\
\approx\ & h(((conv + id) \circ o)^\sharp\ x) && \text{(Preservation)} \\
\approx\ & h\ x. && (*)
\end{aligned}$$

It thus suffices to show that $(e \circ conv + id)(o\ x) \approx (h \circ conv + id)(o\ x)$. Indeed, discriminating over x yields:

Case 1. $x = now\ z$

$$\begin{aligned}
&(e \circ conv + id)(o(now\ z)) \\
\overset{+}{=}\ &(e \circ conv + id)(i_1(now\ z)) \\
\overset{+}{=}\ &e(now\ z) \\
\overset{+}{=}\ &f\ z \\
\overset{+}{=}\ &h(now\ z) \\
\overset{+}{=}\ &(h \circ conv + id)(i_1(now\ z)) \\
\overset{+}{=}\ &(h \circ conv + id)(o(now\ z))
\end{aligned}$$

Case 2. $x = later\ z$

$$\begin{aligned}
&(e \circ conv + id)(o(later\ z)) \\
\overset{+}{=}\ &(e \circ conv + id)(i_2\ z) \\
\overset{+}{=}\ &i_2\ z \\
\overset{+}{=}\ &(h \circ conv + id)(i_2\ z) \\
\overset{+}{=}\ &(h \circ conv + id)(o(later\ z))
\end{aligned}$$

It has thus been proven that every $\tilde{\tilde{D}}\ A$ admits a free Elgot algebra structure. □

Let us now establish some functions for inspecting and manipulating the computation of elements of $D\ A$. These functions and some key facts will then be used to finish the remaining proof needed for Theorem 6.10.

First, consider the ordering with respect to execution time on elements of $D\ A$, defined by

$$\frac{p : x \downarrow a}{now_{\lesssim}\ p : now\ a \lesssim x} \qquad \frac{p : x \lesssim y}{later_{\lesssim}\ p : later\ x \lesssim later\ y}\ .$$

Note that $x \lesssim y$ implies $x \approx y$ for any $x, y : D\ A$, which follows easily by coinduction.

Now, consider the following function $race : D\ A \to D\ A \to D\ A$ which tries running two computations and returns the one that finished first:

$$race\ p\ q := \begin{cases} now\ a & \text{if } p = now\ a \\ now\ b & \text{if } p = later\ a \text{ and } q = now\ b \\ later\ (race\ a\ b) & \text{if } p = later\ a \text{ and } q = later\ b \end{cases}$$

The following Corollary, whose proof can be found in the formalization, will be needed.

Corollary 6.11. *race satisfies the following properties:*

$$\begin{aligned}
&x \approx y \text{ implies } race\ x\ y \sim race\ y\ x \quad \text{for any } x, y : D\ A \\
&x \approx y \text{ implies } race\ x\ y \lesssim y \qquad\quad \text{for any } x, y : D\ A.
\end{aligned}$$

Next, let us consider functions for counting steps of computations, first regard $\Delta_0 : (x : D\ A) \to (a : A) \to (x \downarrow a) \to \mathbb{N}$, which returns the number of steps a terminating computation has to take and is defined by

$$\Delta_0\ x\ a\ p := \begin{cases} 0 & \text{if } x = now\ y \\ (\Delta_0\ y\ a\ q) + 1 & \text{if } x = later\ y \text{ and } p = later_{\downarrow} q \end{cases}$$

Similarly, consider $\Delta : (x, y : D\ A) \to x \lesssim y \to D(A \times \mathbb{N})$ defined by

$$\Delta\ x\ y\ p := \begin{cases} now(a, \Delta_0\ x\ a\ q) & \text{if } x = now\ a \text{ and } p = now_{\lesssim} q \\ later(\Delta\ a\ b\ q) & \text{if } x = later\ a, y = later\ b \text{ and } p = later_{\lesssim} q \end{cases}$$

Lastly, consider the function $\iota : A \times \mathbb{N} \to D\ A$, which adds a number of *later* constructors in front of a value and is given by

$$\iota\ (a, n) := \begin{cases} now\ x & \text{if } n = 0 \\ later(\iota\ (a, m)) & \text{if } n = m + 1 \end{cases}$$

Trivially, ι extends to a setoid morphism $\iota : A \times \mathbb{N} \rightsquigarrow \tilde{D}\ A$, where the equivalence on $\mathbb{N}$ is propositional equality. Let us state two facts about Δ, the proofs can again be found in the formalization.

Corollary 6.12. *Δ satisfies the following properties:*

$$p : x \lesssim y \text{ implies } \tilde{D}(fst(\Delta\ x\ y\ p)) \sim x \text{ for any } x, y : D\ A \tag{Δ_1}$$
$$p : x \lesssim y \text{ implies } \iota^*(\Delta\ x\ y\ p) \sim y \text{ for any } x, y : D\ A. \tag{Δ_2}$$

Let us now return to the missing Corollary of Theorem 6.10.

Corollary 6.13. *The setoid morphism $g^{\sharp_b} : \tilde{D}\ A \rightsquigarrow B$ defined in Theorem 6.10 respects weak bisimilarity, thus yielding $f^\star = g^{\sharp_b} : \tilde{\tilde{D}}\ A \rightsquigarrow B$.*

Proof. Let $x, y : D\ A$ such that $x \approx y$. Recall that by Corollary 6.11 $x \approx y$ implies $p : race\ x\ y \lesssim y$ and symmetrically $q : race\ y\ x \lesssim x$, now, using Corollaries 6.11 and 6.12:

$$\begin{aligned}
& g^{\sharp_b}\ x \\
\overset{B}{=}\ & g^{\sharp_b}(\iota^*(\Delta\ (race\ y\ x)\ x\ q)) && (\Delta_2) \\
\overset{B}{=}\ & g^{\sharp_b}(\tilde{D}fst(\Delta\ (race\ y\ x)\ x\ q)) && (*) \\
\overset{B}{=}\ & g^{\sharp_b}(race\ y\ x) && (\Delta_1) \\
\overset{B}{=}\ & g^{\sharp_b}(race\ x\ y) && \text{(Corollary 6.11)} \\
\overset{B}{=}\ & g^{\sharp_b}(\tilde{D}fst(\Delta\ (race\ x\ y)\ y\ p)) && (\Delta_1) \\
\overset{B}{=}\ & g^{\sharp_b}(\iota^*(\Delta\ (race\ x\ y)\ y\ p)) && (*) \\
\overset{B}{=}\ & g^{\sharp_b}\ y. && (\Delta_2)
\end{aligned}$$

We have thus reduced the proof to showing that

$$g^{\sharp_b}(\tilde{D}fst\ z) \overset{B}{=} g^{\sharp_b}(\iota^*\ z) \text{ for any } z : D(A \times \mathbb{N}). \tag{$*$}$$

Let us proceed as follows

$$\begin{aligned}
& g^{\sharp_b}(\tilde{D}fst\ z) \\
\overset{B}{=}\ & g_1^{\sharp_b}\ z && \textbf{(Uniformity)} \\
\overset{B}{=}\ & g_2^{\sharp_b}\ z \\
\overset{B}{=}\ & g^{\sharp_b}(\iota^*\ z). && \textbf{(Uniformity)}
\end{aligned}$$

Which leaves us to find suitable $g_1, g_2 : \tilde{D}(A \times \mathbb{N}) \rightsquigarrow B + \tilde{D}(A \times \mathbb{N})$. Consider,

$$g_1\ p := \begin{cases} i_1(f\ x) & \text{if } p = now\ (x, zero) \\ i_2(\tilde{D}o(\iota\ (x, n))) & \text{if } p = now\ (x, n+1) \\ i_2\ q & \text{if } p = later\ q \end{cases}$$

and

$$g_2\ p := \begin{cases} i_1(f\ x) & \text{if } p = now\ (x, n) \\ i_2\ q & \text{if } p = later\ q \end{cases}$$

where $o : A \rightsquigarrow A \times \mathbb{N}$ is a setoid morphism that maps every $z : A$ to $(z, 0) : A \times \mathbb{N}$. The applications of **Uniformity** are then justified by the definitions of g_1 and g_2 as well as the fact that $\iota \circ o \doteq now$.

We are thus done after showing that $g_1^{\sharp_b}\ z \overset{B}{=} g_2^{\sharp_b}\ z$. Consider another setoid morphism

$$g_3 : \tilde{D}(A \times \mathbb{N}) \rightsquigarrow B + \tilde{D}(A \times \mathbb{N}) + \tilde{D}(A \times \mathbb{N}),$$

defined by

$$g_3\ p := \begin{cases} i_1(f\ x) & \text{if } p = now\ (x, 0) \\ i_2(i_1(\tilde{D}o(\iota\ (x, n)))) & \text{if } p = now\ (x, n+1) \\ i_2(i_2\ q) & \text{if } p = later\ q \end{cases}$$

Let us now proceed by

$$\begin{aligned}
& g_1^{\sharp_b}\ z \\
\overset{B}{=}\ & ((id + [id, id]) \circ g_3)^{\sharp_b}\ z \\
\overset{B}{=}\ & ([i_1, ((id + [id, id]) \circ g_3)^{\sharp_b} + id] \circ g_3)^{\sharp_b}\ z && \textbf{(Diamond)} \\
\overset{B}{=}\ & g_2^{\sharp_b}\ z.
\end{aligned}$$

Where for the first step notice that $g_1\ x \overset{+}{=} (id + [id, id])(g_3\ x)$ for any $x : \tilde{D}(A \times \mathbb{N})$ follows simply by case distinction on x. For the last step, it suffices to show that $[i_1, ((id + [id, id]) \circ g_3)^{\sharp_b} + id](g_3\ x) \overset{+}{=} g_2\ x$ for any $x : \tilde{D}(A \times \mathbb{N})$. We proceed by case distinction on x.

Case 1. $x = now\ (y, 0)$
The goal reduces to

$$\begin{aligned}
& [i_1, ((id + [id, id]) \circ g_3)^{\sharp_b} + id](g_3\ x) \\
\overset{+}{=}\ & [i_1, ((id + [id, id]) \circ g_3)^{\sharp_b} + id](i_1(f\ y)) \\
\overset{+}{=}\ & i_1(f\ y) \\
\overset{+}{=}\ & g_2\ x,
\end{aligned}$$

which indeed holds by the definitions of g_2 and g_3.

Case 2. $x = now\ (y, n+1)$
The goal reduces to

$$\begin{aligned}
&[i_1, ((id + [id, id]) \circ g_3)^{\sharp_b} + id](g_3\ x) \\
\stackrel{+}{=}\ &[i_1, ((id + [id, id]) \circ g_3)^{\sharp_b} + id](i_2(i_1(\tilde{Do}(\iota\ (y, n))))) \\
\stackrel{+}{=}\ &i_1(((id + [id, id]) \circ g_3)^{\sharp_b}((\tilde{Do}(\iota\ (y, n))))) \\
\stackrel{+}{=}\ &i_1(f\ y) && (*) \\
\stackrel{+}{=}\ &g_2\ x
\end{aligned}$$

Where

$$((id + [id, id]) \circ g_3)^{\sharp_b}(\tilde{Do}(\iota\ (y, n))) \stackrel{B}{=} f\ y \qquad (*)$$

follows by induction on n:

Subcase (i): $n = 0$
We are done by

$$\begin{aligned}
&((id + [id, id]) \circ g_3)^{\sharp_b}(\tilde{Do}(\iota\ (y, 0))) \\
\stackrel{B}{=}\ &((id + [id, id]) \circ g_3)^{\sharp_b}(\tilde{Do}(now\ y)) \\
\stackrel{B}{=}\ &((id + [id, id]) \circ g_3)^{\sharp_b}(now(y, 0)) \\
\stackrel{B}{=}\ &([id, ((id + [id, id]) \circ g_3)^{\sharp_b}] \circ (id + [id, id]) \circ g_3)(now(y, 0)) && (\textbf{Fixpoint}) \\
\stackrel{B}{=}\ &([id, ((id + [id, id]) \circ g_3)^{\sharp_b}] \circ (id + [id, id]))i_1(f\ y) \\
\stackrel{B}{=}\ &[id, ((id + [id, id]) \circ g_3)^{\sharp_b}]i_1(f\ y) \\
\stackrel{B}{=}\ &f\ y
\end{aligned}$$

Subcase (ii): $n = m + 1$
Assuming that $((id + [id, id]) \circ g_3)^{\sharp_b}(\tilde{Do}(\iota\ (y, m))) \stackrel{B}{=} f\ y$, we are done by

$$\begin{aligned}
&((id + [id, id]) \circ g_3)^{\sharp_b}(\tilde{Do}(\iota\ (y, m+1))) \\
\stackrel{B}{=}\ &((id + [id, id]) \circ g_3)^{\sharp_b}(\tilde{Do}(later(\iota\ (y, m)))) \\
\stackrel{B}{=}\ &((id + [id, id]) \circ g_3)^{\sharp_b}(later(\tilde{Do}(\iota\ (y, m)))) \\
\stackrel{B}{=}\ &([id, ((id + [id, id]) \circ g_3)^{\sharp_b}] \circ (id + [id, id]) \circ g_3)(later(\tilde{Do}(\iota\ (y, m)))) && (\textbf{Fixpoint}) \\
\stackrel{B}{=}\ &([id, ((id + [id, id]) \circ g_3)^{\sharp_b}] \circ (id + [id, id]))(i_2(i_2(\tilde{Do}(\iota\ (y, m))))) \\
\stackrel{B}{=}\ &[id, ((id + [id, id]) \circ g_3)^{\sharp_b}](\tilde{Do}(\iota\ (y, m))) \\
\stackrel{B}{=}\ &f\ y
\end{aligned}$$

Case 3. $x = later\ p$
The goal reduces to

$$\begin{aligned}
&[i_1, ((id + [id, id]) \circ g_3)^{\sharp_b} + id](g_3\ x) \\
\stackrel{+}{=}\ &[i_1, ((id + [id, id]) \circ g_3)^{\sharp_b} + id](i_2(i_2\ p)) \\
\stackrel{+}{=}\ &i_2\ p \\
\stackrel{+}{=}\ &g_2\ x,
\end{aligned}$$

which instantly follows by definition.

This finishes the proof of the Corollary and thus Theorem 6.10 holds. □

We have shown in Theorem 6.10 that every $\tilde{\tilde{D}}\ A$ extends to a free Elgot algebra. Together with Proposition 6.5 and Lemma 5.15 this yields a description for the monad $\mathbf{K}$ which has been defined in Chapter 5, in the category *Setoids*.

7 Conclusion

We have considered a novel approach to defining a monad suitable for modelling partiality from first principles, which has first been introduced in [19]. Using the dependently typed programming language Agda, we were able to formally verify important properties of this monad: it is an equational lifting monad, i.e. a monad that offers no other side effect besides some form of non-termination and furthermore it turns out to be the initial pre-Elgot monad. Moreover, we have considered a concrete description of this monad in the category of setoids, where it turns out to be a quotient of the delay monad.

With this thesis we have thus created a small Agda library that contains categorical concepts concerning partiality and iteration theories. Future work might improve on this library by formalizing important results concerning partiality monads, such as the fact that every equational lifting monad has a restriction category as its Kleisli category. Furthermore, one can continue studying the delay monad in a categorical setting, by modeling the quotient by weak bisimilarity of the delay monad through a certain coequalizer, as has been done in [19], and then identifying assumptions under which this constitutes a suitable monad for modeling partiality.

Bibliography

[1] J. Lambek, 'A fixpoint theorem for complete categories,' *Mathematische Zeitschrift*, vol. 103, pp. 151–161, 1968.

[2] S. M. Lane, 'Categories for the working mathematician,' 1971. [Online]. Available: https://api.semanticscholar.org/CorpusID:122892655.

[3] G. D. Plotkin, 'Call-by-name, call-by-value and the λ-calculus,' *Theoretical computer science*, vol. 1, no. 2, pp. 125–159, 1975.

[4] E. G. Manes, 'Algebraic theories in a category,' *Algebraic Theories*, pp. 161–279, 1976.

[5] E. Moggi, 'Notions of computation and monads,' *Inf. Comput.*, vol. 93, no. 1, pp. 55–92, Jul. 1991, ISSN: 0890-5401. DOI: 10.1016/0890-5401(91)90052-4. [Online]. Available: https://doi.org/10.1016/0890-5401(91)90052-4.

[6] D. S. Scott, 'A type-theoretical alternative to iswim, cuch, owhy,' *Theoretical Computer Science*, vol. 121, no. 1-2, pp. 411–440, 1993.

[7] V. Vene, *Categorical programming with inductive and coinductive types*. Citeseer, 2000.

[8] L. S. Moss, 'Parametric corecursion,' *Theoretical Computer Science*, vol. 260, no. 1, pp. 139–163, 2001, Coalgebraic Methods in Computer Science 1998, ISSN: 0304-3975. DOI: https://doi.org/10.1016/S0304-3975(00)00126-2. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0304397500001262.

[9] J. R. B. Cockett and S. Lack, 'Restriction categories i: Categories of partial maps,' *Theor. Comput. Sci.*, vol. 270, no. 1–2, pp. 223–259, Jan. 2002, ISSN: 0304-3975. DOI: 10.1016/S0304-3975(00)00382-0. [Online]. Available: https://doi.org/10.1016/S0304-3975(00)00382-0.

[10] P. Aczel, J. Adámek, S. Milius, and J. Velebil, 'Infinite trees and completely iterative theories: A coalgebraic view,' *Theor. Comput. Sci.*, vol. 300, no. 1–3, pp. 1–45, May 2003, ISSN: 0304-3975. DOI: 10.1016/S0304-3975(02)00728-4. [Online]. Available: https://doi.org/10.1016/S0304-3975(02)00728-4.

[11] A. Bucalo, C. Führmann, and A. Simpson, 'An equational notion of lifting monad,' *Theor. Comput. Sci.*, vol. 294, no. 1–2, pp. 31–60, Feb. 2003, ISSN: 0304-3975. DOI: 10.1016/S0304-3975(01)00243-2. [Online]. Available: https://doi.org/10.1016/S0304-3975(01)00243-2.

[12] V. Capretta, 'General recursion via coinductive types,' *CoRR*, vol. abs/cs/0505037, 2005. arXiv: cs/0505037. [Online]. Available: http://arxiv.org/abs/cs/0505037.

[13] J. Adámek, S. Milius, and J. Velebil, 'Elgot algebras,' *CoRR*, vol. abs/cs/0609040, 2006. arXiv: cs/0609040. [Online]. Available: http://arxiv.org/abs/cs/0609040.

[14] J. Adámek, S. Milius, and J. Velebil, 'Elgot theories: A new perspective on the equational properties of iteration,' *Mathematical Structures in Computer Science*, vol. 21, no. 2, pp. 417–480, 2011. DOI: 10.1017/S0960129510000496.

[15] Y. Kinoshita and J. Power, 'Category theoretic structure of setoids,' *Theoretical Computer Science*, vol. 546, pp. 145–163, 2014, Models of Interaction: Essays in Honour of Glynn Winskel, ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2014.03.006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0304397514001819.

[16] J. Chapman, T. Uustalu, and N. Veltri, 'Quotienting the delay monad by weak bisimilarity,' in *Proceedings of the 12th International Colloquium on Theoretical Aspects of Computing - ICTAC 2015 - Volume 9399*, Berlin, Heidelberg: Springer-Verlag, 2015, pp. 110–125, ISBN: 9783319251493. DOI: 10.1007/978-3-319-25150-9_8. [Online]. Available: https://doi.org/10.1007/978-3-319-25150-9_8.

[17] S. Goncharov, L. Schröder, C. Rauch, and M. Piróg, 'Unifying guarded and unguarded iteration,' in *Foundations of Software Science and Computation Structures: 20th International Conference, FOSSACS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings 20*, Springer, 2017, pp. 517–533.

[18] S. Goncharov, L. Schröder, C. Rauch, and J. Jakob, 'Unguarded recursion on coinductive resumptions,' *Logical Methods in Computer Science*, vol. 14, 2018.

[19] S. Goncharov, 'Uniform elgot iteration in foundations,' *CoRR*, vol. abs/2102.11828, 2021. arXiv: 2102.11828. [Online]. Available: https://arxiv.org/abs/2102.11828.

[20] J. Z. S. Hu and J. Carette, 'Formalizing category theory in agda,' in *Proceedings of the 10th ACM SIGPLAN International Conference on Certified Programs and Proofs*, ser. CPP 2021, Virtual, Denmark: Association for Computing Machinery, 2021, pp. 327–342, ISBN: 9781450382991. DOI: 10.1145/3437992.3439922. [Online]. Available: https://doi.org/10.1145/3437992.3439922.

[21] The Agda Community, *Agda Standard Library*, version 2.0, Dec. 2023. [Online]. Available: https://github.com/agda/agda-stdlib.

[22] T. A. Team, *Agda user manual*, version 2.6.4.3, Mar. 2024. [Online]. Available: https://agda.readthedocs.io/en/v2.6.4.3/.

[23] T. C. D. Team, *The coq reference manual*, version 8.19.1, Mar. 2024. [Online]. Available: https://coq.inria.fr/doc/V8.19.0/refman/.

[24] Agda Developers, *Agda*, version 2.6.5. [Online]. Available: https://agda.readthedocs.io/.