Fibrational Induction Meets Effects

Robert Atkey*, Neil Ghani*, Bart Jacobs†, and Patricia Johann*

†Radboud University, The Netherlands, bart@cs.ru.nl
*University of Strathclyde, Scotland
{Robert.Atkey, Neil.Ghani, Patricia.Johann}@cis.strath.ac.uk

Abstract. This paper provides several induction rules that can be used to prove properties of effectful data types. Our results are semantic in nature and build upon Hermida and Jacobs' fibrational formulation of induction for polynomial data types and its extension to all inductive data types by Ghani, Johann, and Fumex. An effectful data type $\mu(TF)$ is built from a functor F that describes data, and a monad T that computes effects. Our main contribution is to derive induction rules that are generic over *all* functors F and monads T such that $\mu(TF)$ exists. Along the way, we also derive a principle of definition by structural recursion for effectful data types that is similarly generic. Our induction rule is also generic over the kinds of properties to be proved: like the work on which we build, we work in a general fibrational setting and so can accommodate very general notions of properties, rather than just those of particular syntactic forms. We give examples exploiting the generality of our results, and show how our results specialize to those in the literature, particularly those of Filinski and Støvring.

1 Introduction

Induction is a powerful principle for proving properties of data types and the programs that manipulate them. Probably the simplest induction rule is the familiar one for the set of natural numbers: For any property P of natural numbers, if $P0$ holds, and if $P(n+1)$ holds whenever Pn holds, then Pn holds for all natural numbers n. As early as the 1960s, Burstall [2] observed that induction rules are definable for various forms of tree-like data types as well. The data types he considered can all be modelled by polynomial functors on Set, and even today induction is most often used to prove properties of these types. But while most treatments of induction for tree-like data types use a specific notion of predicate, other reasonable notions are possible. For example, a predicate on a set A is often taken by type theorists to be a function $P : A \to \mathsf{Set}$, by category theorists to be an object of the slice category Set/A, and by logicians to be a subset of A. Thus, even just for tree-like data types, induction rules are typically derived on an *ad hoc* basis, with seemingly different results available for the different kinds of properties of data types and their programs to be proved.

Until fairly recently a comprehensive and general treatment of induction remained elusive. But in 1998 Hermida and Jacobs [9] showed how to replace *ad hoc* treatments of induction by a unifying axiomatic approach based on fibrations. The use of fibrations was motivated by the facts that i) the semantics of data types in languages involving, say, non-termination usually involves categories other than Set; ii) in such circumstances, standard set-based notions of

predicates are no longer germane; iii) even when working in Set there are many reasonable notions of predicate (e.g., the three mentioned above); and iv) when deriving induction rules for more sophisticated classes of data types, we do not want to have to develop a specialised theory of induction for each one; We hope instead to appropriately instantiate a single, generic, axiomatic theory of induction that is widely applicable and abstracts over the specific choices of category, functor, and predicate giving rise to different induction rules for specific classes of data types. Fibrations support precisely such an axiomatic approach.

Although Hermida and Jacobs derive their induction rules only for data types modelled by polynomial functors, this result was recently extended to all inductive data types — i.e., all data types that are fixed points of functors — by Ghani, Johann, and Fumex [7,8]. Examples of non-polynomial inductive data types include rose trees, inductive families (e.g., perfect trees), quotient types (e.g., finite power sets), and hyperfunctions. These data types are sophisticated, but nevertheless are still pure, i.e., effect-free. This leads us to ask:

How can we reason inductively about data types in the presence of effects?

In this situation, we are interested in *effectful data structures* — i.e., data structures whose constructors can perform effectful computations — and the (possibly effectful) programs that manipulate them. Such programs can fail to terminate, raise exceptions, alter state, perform non-deterministic computations, and so on.

Moggi's landmark paper [13] suggests that one way to handle effectful programs is to model effects via a monad T, where TX represents effectful computations that return values of type X with effects described by T. In Haskell, for example, input/output effects are modelled by the monad `IO`, and we can define the following effectful data type of IO-lists:

```
type IOList a  = IO (IOList' a)

data IOList' a = IONil | IOCons a (IO (IOList' a))
```

For any list of type `IOList a`, some `IO` action must be performed to discover whether or not there is an element at the head of the list. Additional IO actions must be performed to obtain any remaining elements. Such a data type could be used to read list data "on demand" from some file or input device, for instance. Recalling that the standard append function is associative on pure lists, and observing that standard induction techniques for lists do not apply to functions on effectful data types, we can ask whether or not it is possible to prove by induction that the following effectful append function is associative on IO-lists:

```
appIO :: IOList a -> IOList a -> IOList a
appIO s t = do z <- s
               case z of IONil       -> t
                         IOCons w u -> return (IOCons w (appIO u t))
```

In fact, an even more fundamental question must first be addressed: How do we know that `appIO` is well-defined? After all, it is not at all obvious that the argument `u` to the recursive call of `appIO` is smaller than the original input `s`.

More generally, we can consider effectful data types given by the following type definitions. These generalise IO-lists by abstracting, via `f`, over the data structure involved and, via `m`, over the monad involved.

```
type D f m = m (Mu f m)

data Mu f m = In (f (m (Mu f m))
```

We can then ask whether structural recursion can be used to define functions on data structures of type `D f m` and induction can be used to prove their properties.

Filinski and Støvring [5] provide partial answers to these questions. Taking types to be interpreted in the category CPO of ω-complete partial orders and total continuous functions, and taking a predicate to be an admissible subset of a CPO, they give a mathematically principled induction rule for establishing the truth of predicates for effectful strictly positive data types that can be modelled in CPO. Their induction rule is modular, in that they separate the premises for inductive reasoning about data structures from those for inductive reasoning about effects, and a number of examples are given to illustrate its use. Filinski and Støvring also give a principle of definition by structural recursion for effectful data types. But because they restrict attention to CPO, to a syntactically restricted class of functors, and to a particular notion of predicate, their results are not as widely applicable as we might hope.

In this paper we show how the fibrational approach to induction can be extended to the effectful setting. We obtain a generalisation of Filinski and Støvring's induction rule that is completely free of the above three restrictions. We also derive a principle of definition by structural recursion for effectful data types that is similarly restriction-free. Our principle of definition can be used to show that `appIO` is well-defined, and our induction rule can be used to show that it is associative on IO-lists (see Example 6). These results lie outside the scope of Filinski and Støvring's work. (Interestingly, while the standard reverse function is an involution on lists, a similarly effectful reverse function is *not* an involution on IO-lists, so not all results transfer from the pure setting to the effectful one.) When specialised to the fibration of subobjects of CPO implicitly used by Filinski and Støvring, Theorem 2 and Corollary 1 give precisely their definition principle and induction rule, respectively. But because we treat functors that are not strictly positive, we are able to derive results for data types they cannot handle. Moreover, even if we restrict to the same class of functors as Filinski and Støvring, our fibrational approach allows us to derive, in addition to our generalisation of their modular one, another more powerful effectful induction rule (see Theorem 4). More specifically, our contributions are as follows:

- Given a functor F and a monad T, we first show in Theorem 2 that the carrier $\mu(TF)$ of the initial TF-algebra deserves to be thought of as the effectful data type built from data described by F and effects computed by T. In fact, the effectful data types introduced by the Haskell code above are all of the form $\mu(TF)$. Informally, the data type $\mu(TF)$ contains all interleavings of F and T; formally, it is the free structure which is the carrier of both

an F-algebra and Eilenberg-Moore algebra for T. Theorem 2 also gives a principle of definition by structural recursion for effectful data types.
- We then turn to the question of proof by induction and show that there are a number of useful induction rules for effectful data types. (See Corollary 1 and Theorems 4, 6, and 7.) We consider the relative merits of these rules, and note that in the proof-irrelevant case, which includes that considered by Filinski and Støvring, Theorems 4 and 6 coincide. We note that each of our induction rules also gives us a definitional format for dependently typed functions whose domains are effectful data types. This generalises the elimination rules for (pure) inductive data types in Martin-Löf Type Theory.
- Finally, we consider effectful induction in fibrations having very strong sums. Examples include the codomain and families fibrations, used heavily by category theorists and type theorists, respectively. In such fibrations, we show that the key operation of *lifting* is a strong monoidal functor from the category of endofunctors on the base category to the category of endofunctors on the total category of the fibration, and thus that liftings preserve monads. This ensures that all of our inductive reasoning can be performed in the total category of the fibration (see Section 6).

The rest of this paper is structured as follows. Section 2 introduces some categorical preliminaries. Section 3 formalises the notion of an effectful data type. It also shows how to construct algebras for the composition of two functors from algebras for each of the component functors, gives a converse construction when one of the functors is a monad, and uses these two constructions to generalise Filinski and Støvring's induction rule to arbitrary effectful data types. This is all achieved without the use of fibrations, although, implicitly, Section 3 takes place entirely within the subobject fibration over CPO. Section 4 reviews the fibrational approach to induction. Section 5 relaxes the restriction to the subobject fibration on CPOs and uses arbitrary fibrations explicitly to further abstract the notion of predicate under consideration. We capitalise on this abstraction to give a number of different induction rules derivable in the fibrational setting. Section 6 considers induction in the presence of very strong sums. Section 7 concludes and discuss directions for further research.

2 Categorical Preliminaries

We assume familiarity with basic category theory, initial algebra semantics of data types, and the representation of computational effects as monads.

Let $\mathcal{B}$ be a category and $F : \mathcal{B} \to \mathcal{B}$ be a functor. Recall that an *F-algebra* is a morphism $h : FX \to X$ for some object X of $\mathcal{B}$, called the *carrier* of h. For any functor F, the collection of F-algebras itself forms the category Alg_F. In Alg_F, an *F-algebra morphism* from the F-algebra $h : FX \to X$ to the F-algebra $g : FY \to Y$ is a morphism $f : X \to Y$ such that $f \circ h = g \circ Ff$. When it exists, the initial F-algebra $in : F(\mu F) \to \mu F$ is unique. We write U_F for the *forgetful functor* mapping each F-algebra to its carrier, and we suppress the subscript F on U and on in when convenient.

Let $\mathcal{B}$ be a category and (T, η, μ) be a monad on $\mathcal{B}$.[1] We write T for the monad (T, η, μ) when no confusion may result. An *Eilenberg-Moore algebra for* T is a T-algebra $h : TX \to X$ such that $h \circ \mu_X = h \circ Th$ and $h \circ \eta_X = id$; we can think of such algebra as a T-algebra that respects the unit and multiplication of T. The collection of Eilenberg-Moore algebras for a monad T forms the category EM_T. In EM_T, an EM_T*-morphism* from an Eilenberg-Moore algebra $h : TX \to X$ to an Eilenberg-Moore algebra $g : TY \to Y$ is just a T-algebra morphism from h to g. It is easy to check that EM_T is a full subcategory of Alg_T.

3 Effectful Data Types and an Effectful Induction Rule

The carrier μF of the initial F-algebra can thought of as the data type defined by F. But if we are in an effectful setting, with effects modelled by a monad (T, η, μ), then what type should we consider the effectful data type defined by F and T together? Whatever it is, it should be the carrier of an F-algebra f that describes the data. It should also be the carrier of a Eilenberg-Moore algebra g for T so that it respects η and μ. Finally, it should be the carrier of an algebra constructed from f and g that is initial in the same way that μF is the carrier of the initial F-algebra. We therefore define the effectful data type generated by F and T to be $\mu(TF)$. Moreover, $in \circ \eta$ is an F-algebra, and $in \circ \mu_T \circ T(in^{-1})$ is an Eilenberg-Moore algebra for T, both with carrier $\mu(TF)$. It is easy to verify that no T-algebra structure exists on the other obvious choice, namely $\mu(FT)$.

Unfortunately, however, carriers of initial TF-algebras can be hard to work with. We therefore write $\mu(TF)$ for $T(\mu(FT))$ when convenient. This is justified by the "rolling lemma" [6], which entails that if F and G are functors such that $\mu(GF)$ exists, then $\mu(FG)$ exists and $\mu(GF) = G(\mu(FG))$. The data types `D f m` from the introduction all have the form $T(\mu(FT))$. We establish that $T(\mu(FT))$ satisfies the above specification by first showing in Lemma 1 how to construct FT-algebras from F-algebras and T-algebras, and then refining this construction in Theorem 1 to take into account that T is a monad and we are actually interested in its Eilenberg-Moore algebras. We begin with a definition.

Definition 1 *Let $F, G : \mathcal{B} \to \mathcal{B}$ be functors, and let $\mathsf{Alg}_F \times_{\mathcal{B}} \mathsf{Alg}_G$ be defined by the pullback of $U_F : \mathsf{Alg}_F \to \mathcal{B}$ and $U_G : \mathsf{Alg}_G \to \mathcal{B}$ in Cat. An F-and-G-algebra is an object of $\mathsf{Alg}_F \times_{\mathcal{B}} \mathsf{Alg}_G$, i.e., a triple comprising an object A of $\mathcal{B}$, an F-algebra $f : FA \to A$, and a G-algebra $g : GA \to A$. Morphisms of $\mathsf{Alg}_F \times_{\mathcal{B}} \mathsf{Alg}_G$ are morphisms of $\mathcal{B}$ that are simultaneously F-algebra and G-algebra morphisms.*

Lemma 1. *Let $F, G : \mathcal{B} \to \mathcal{B}$ be functors. There is a functor $\Phi : \mathsf{Alg}_F \times_{\mathcal{B}} \mathsf{Alg}_G \to \mathsf{Alg}_{FG}$ that sends F-and-G-algebras to FG-algebras.*

Proof. Define $\Phi(A, f, g) = f \circ Fg$. The action of Φ on morphisms is obvious. □

In the setting of effectful data types, where G is a monad in whose Eilenberg-Moore algebras we are interested, Lemma 1 can be strengthened.

[1] We use μ to denote both least fixed points of functors and multiplication operators of monads as is traditional. Which is meant when will be made clear from context.

Definition 2 *Let $F : \mathcal{B} \to \mathcal{B}$ be a functor, let (T, η, μ) be a monad on $\mathcal{B}$, and let $\mathsf{Alg}_F \times_B \mathsf{EM}_T$ be the category defined by the pullback of $U_F : \mathsf{Alg}_F \to \mathcal{B}$ and $U_T : \mathsf{EM}_T \to \mathcal{B}$ in Cat. An F-and-T-*Eilenberg-Moore algebra *is an object of $\mathsf{Alg}_F \times_B \mathsf{EM}_T$, i.e., a triple comprising an object A of $\mathcal{B}$, an F-algebra $f : FA \to A$, and an Eilenberg-Moore algebra $g : TA \to A$ for T. Morphisms of $\mathsf{Alg}_F \times_B \mathsf{EM}_T$ are morphisms of $\mathcal{B}$ that are simultaneously F-algebra morphisms and EM_T-morphisms, i.e., F-algebra morphisms and T-algebra morphisms.*

The key point about F-and-T-Eilenberg-Moore algebras is that the functor mapping them to FT-algebras has a left adjoint. Since every Eilenberg-Moore algebra for T is a T-algebra, we abuse notation and also call this functor Φ.

Theorem 1. *Let $F : \mathcal{B} \to \mathcal{B}$ be a functor and (T, η, μ) be a monad on $\mathcal{B}$. The functor $\Phi : \mathsf{Alg}_F \times_{\mathcal{B}} \mathsf{EM}_T \to \mathsf{Alg}_{FT}$ has a left adjoint $\Psi : \mathsf{Alg}_{FT} \to \mathsf{Alg}_F \times_{\mathcal{B}} \mathsf{EM}_T$.*

Proof. Define Ψ by $\Psi(k : FTA \to A) = (TA, \eta \circ k : FTA \to TA, \mu : T^2A \to TA)$ on objects and by $\Psi f = Tf$ on morphisms. For any FT-algebra morphism $f : A \to B$, naturality of η and μ ensure that Ψf is a morphism in $\mathsf{Alg}_F \times_{\mathcal{B}} \mathsf{EM}_T$.

To see that Φ and Ψ are adjoint let $k : FTA \to A$ be an object of Alg_{FT} and $(B, f : FB \to B, g : TB \to B)$ be an object of $\mathsf{Alg}_F \times_{\mathcal{B}} \mathsf{EM}_T$. We construct a natural isomorphism between morphisms from $\Psi\, k$ to (B, f, g) and morphisms from k to $\Phi(B, f, g)$. Given $h : \Psi\, k \to (B, f, g)$, define $\phi(h) : k \to \Phi(B, f, g)$ by $\phi(h) = h \circ \eta$. Then $\phi(h)$ is an FT-algebra morphism because $f \circ Fg \circ FT\phi(h) = f \circ Fg \circ FTh \circ FT\eta = f \circ Fh \circ F\mu \circ FT\eta = f \circ Fh = h \circ \eta \circ k = \phi(h) \circ k$. Here, the first equality holds because FT is a functor, the second holds because h is a T-algebra morphism, the third by the monad laws, and the fourth because h is an F-algebra morphism. Conversely, given $h : k \to \Phi(B, f, g)$, define $\psi(h) : \Psi\, k \to (B, f, g)$ by $\psi(h) = g \circ Th$. Then $\psi(h)$ is an F-algebra morphism because $\psi(h) \circ \eta \circ k = g \circ Th \circ \eta \circ k = g \circ \eta \circ h \circ k = h \circ k = f \circ Fg \circ FTh = f \circ F\psi(h)$. Here, the second equality holds by naturality of η, the third because g is an Eilenberg-Moore algebra for T, and the fourth since h is an FT-algebra morphism. Moreover, $\psi(h)$ is a T-algebra morphism because $g \circ T\psi(h) = g \circ Tg \circ TTh = g \circ \mu \circ TTh = g \circ Th \circ \mu = \psi(h) \circ \mu$. Here, the second equality holds since g is an Eilenberg-Moore algebra for T and the third holds by naturality of μ.

To see that ϕ and ψ constitute an isomorphism, first note that $\phi(\psi(h)) = \phi(g \circ Th) = g \circ Th \circ \eta = g \circ \eta \circ h = h$ by naturality of η and the fact that g is an Eilenberg-Moore algebra for T. We also have that $\psi(\phi(h)) = \psi(h \circ \eta) = g \circ Th \circ T\eta = h \circ \mu \circ T\eta = h$ by the fact that h is a T-algebra morphism and the monad laws. Naturality of ϕ and ψ is easily checked. □

A slightly slicker proof abstracts away from the category of Eilenberg-Moore algebras for T to any adjunction $L \dashv R : \mathcal{B} \to \mathcal{D}$ whose induced monad RL is T. In this setting, we define $\mathsf{Alg}_F \times_{\mathcal{B}} \mathcal{D}$ to be the pullback of the forgetful functor U_F and R. The adjunction $L \dashv R : \mathcal{B} \to \mathcal{D}$ then lifts to an adjunction $L^\dagger \dashv R^\dagger : \mathsf{Alg}_{FT} \to \mathsf{Alg}_F \times_{\mathcal{B}} D$. Theorem 1 is the special case of this more general construction for which $\mathcal{D}$ is EM_T. Another special case takes $\mathcal{D}$ to be the Kleisli category of T.

We can now give our principle of definition by effectful structural recursion.

Theorem 2. *Let F be a functor and T be a monad. Then $T(\mu(FT))$, if it exists, is the carrier of the initial F-and-T-Eilenberg-Moore algebra.*

Proof. Since in_{FT} is the initial FT-algebra, and since left adjoints preserve initial objects, we have that the initial F-and-T-Eilenberg-Moore algebra is Ψin_{FT}, i.e., the triple $(T(\mu(FT)), \eta \circ in, \mu)$. □

Given an F-and-T-Eilenberg-Moore algebra (A, f, g), Theorem 2 ensures the existence of a unique F-and-T-Eilenberg-Moore algebra morphism from the initial such algebra to (A, f, g). This gives a morphism from $T(\mu(FT))$ to A, and hence a principle of definition by effectful structural recursion. Indeed, when T is the identity monad, we recover precisely the standard principle of definition by structural recursion for (pure) carriers of initial algebras.

Example 1. We can place the definition of `appIO` from the introduction on a formal footing as follows. First note that `IOList a` is of the form $T(\mu(FT))$, where $FX = 1 + a \times X$ and T is the monad `IO`. The F-and-T-Eilenberg-Moore algebra whose F-algebra sends $inl\ *$ to ys and $inr(z, zs)$ to $(\eta \circ in)(inr(z, zs))$, and whose Eilenberg-Moore algebra for T is μ, defines `appIO _ ys`. It further ensures that `appIO _ ys` is a T-algebra morphism between the T-algebra structure within the initial F-and-T-Eilenberg-Moore algebra and the T-algebra structure within the F-and-T-Eilenberg-Moore algebra just defined.

From Theorem 2 we also get the first of our effectful induction rules.

Corollary 1. *Let P be a subobject of $T(\mu(FT))$, as well as the carrier of an F-and-T-Eilenberg-Moore algebra such that the inclusion map from P to $T(\mu(FT))$ is an F-and-T-Eilenberg-Moore algebra morphism. Then $P = T(\mu(FT))$.*

Proof. There is an F-and-T-Eilenberg-Moore algebra morphism from the initial such algebra to any with carrier P, and this induces a morphism from $T(\mu(FT))$ to P. That this morphism is an inverse to the inclusion map from P to $T(\mu(FT))$ follows from initiality and the fact that the inclusion map is monic. □

In Theorem 7 below we generalise Corollary 1 to handle more general notions of predicate than that given by subobjects. But first we argue that Corollary 1 specialises to recover Filinski and Støvring's induction rule. This rule assumes a minimal T-invariant (TC, i) for F and a subset P of TC that is both T-admissible and F-closed, and concludes that $P = TC$. But i) the minimal T-invariant for F is precisely the initial F-and-T-algebra $T(\mu(FT))$ in CPO, ii) $P \subseteq T(\mu(FT))$ is T-admissible iff there exists a T-algebra $k : TP \to P$ such that the inclusion map ι from P to $T(\mu(FT))$ is a T-algebra morphism from k to $\mu : T^2(\mu(FT)) \to T(\mu(FT))$, and iii) $P \subseteq T(\mu(FT))$ is F-closed iff there exists an F-algebra $h : FP \to P$ such that ι is an F-algebra morphism from h to $\eta \circ in$. Thus, P is the carrier of an F-and-T-Eilenberg-Moore algebra. Moreover, since k coincides with μ and h coincides with $\eta \circ in$ on P, ι is an F-and-T-Eilenberg-Moore algebra morphism. Thus, by Corollary 1, $P = T(\mu(FT))$. Of course this observation allows us to handle all of Filinski and Støvring's examples.

4 Induction in a Fibrational Setting

Thus far we have characterised effectful data types and given our first induction rule for them. This rule is generic over the category interpreting data types, as well as over both the monad interpreting the effects in question and the functor constructing the data type, and specialises to Filinski and Støvring's rule. On the other hand, it holds only for a specific notion of predicate, namely that given by subobjects. Since we seek an induction rule that is also generic over predicates, we turn to fibrations, which support an axiomatic approach to them. We begin by recalling the basics of fibrations. More details can be found in, e.g., [10].

Let $U : \mathcal{E} \to \mathcal{B}$ be a functor. A morphism $g : Q \to P$ in $\mathcal{E}$ is *cartesian* over a morphism $f : X \to Y$ in $\mathcal{B}$ if $Ug = f$, and for every $g' : Q' \to P$ in $\mathcal{E}$ for which $Ug' = f \circ v$ for some $v : UQ' \to X$ there exists a unique $h : Q' \to Q$ in $\mathcal{E}$ such that $Uh = v$ and $g \circ h = g'$. The cartesian morphism $f^{\S}_P$ over a morphism f with codomain UP is unique. We write f^*P for the domain of $f^{\S}_P$.

Cartesian morphisms are the essence of fibrations. A functor $U : \mathcal{E} \to \mathcal{B}$ is a *fibration* if for every object P of $\mathcal{E}$ and every morphism $f : X \to UP$ in $\mathcal{B}$ there is a cartesian morphism $f^{\S}_P : f^*P \to P$ in $\mathcal{E}$ such that $U(f^{\S}_P) = f$. If $U : \mathcal{E} \to \mathcal{B}$ is a fibration, we call $\mathcal{B}$ the *base category* of U and $\mathcal{E}$ the *total category* of U. Objects of $\mathcal{E}$ can be thought of as predicates, objects of $\mathcal{B}$ can be thought of as types, and U can be thought of as mapping each predicate P in $\mathcal{E}$ to the type UP on which P is a predicate. We say that an object P in $\mathcal{E}$ is *over* its image UP under U, and similarly for morphisms. For any object X of $\mathcal{B}$, we write $\mathcal{E}_X$ for the *fibre over* X, i.e., for the subcategory of $\mathcal{E}$ consisting of objects over X and vertical morphisms, i.e., morphisms over id_X. If $f : X \to Y$ is a morphism in $\mathcal{B}$, then the function mapping each object P of $\mathcal{E}$ to f^*P extends to a functor $f^* : \mathcal{E}_Y \to \mathcal{E}_X$. We call the functor f^* the *reindexing functor induced by* f.

Example 2. The category $\mathsf{Fam}(\mathsf{Set})$ has as objects pairs (X, P), where X is a set and $P : X \to \mathsf{Set}$. We refer to (X, P) simply as P when convenient and call X its *domain*. A morphism from $P : X \to \mathsf{Set}$ to $P' : X' \to \mathsf{Set}$ is a pair $(f, f^\sim) : P \to P'$, where $f : X \to X'$ and $f^\sim : \forall x : X.\, P\,x \to P'(f\,x)$. The functor $U : \mathsf{Fam}(\mathsf{Set}) \to \mathsf{Set}$ mapping (X, P) to X is called the *families fibration*.

Dependently typed programmers typically work in the families fibration, in which induction amounts to defining dependently typed functions. It can be generalised (in an equivalent form) to the following fibration.

Example 3. Let $\mathcal{B}$ be a category. The *arrow category* of $\mathcal{B}$, denoted $\mathcal{B}^{\to}$, has the morphisms of $\mathcal{B}$ as its objects. A morphism in $\mathcal{B}^{\to}$ from $f : X \to Y$ to $f' : X' \to Y'$ is a pair (α_1, α_2) of morphisms in $\mathcal{B}$ such that $f' \circ \alpha_1 = \alpha_2 \circ f$. The codomain functor $cod : \mathcal{B}^{\to} \to \mathcal{B}$ maps an object $f : X \to Y$ of $\mathcal{B}^{\to}$ to the object Y of $\mathcal{B}$. If $\mathcal{B}$ has pullbacks, then cod is a fibration, called the *codomain fibration over* $\mathcal{B}$: given an object $f : X \to Y$ in $(\mathcal{B}^{\to})_Y$ and a morphism $f' : X' \to Y$ in $\mathcal{B}$, the pullback of f and f' gives a cartesian morphism over f at f'.

Example 4. If $\mathcal{B}$ is a category, then the category of subobjects of $\mathcal{B}$, denoted $Sub(\mathcal{B})$, has (equivalence classes of) monomorphisms in $\mathcal{B}$ as its objects. A

monomorphism $f : X \hookrightarrow Y$ is called a *subobject* of Y. A morphism in $Sub(\mathcal{B})$ from $f : X \hookrightarrow Y$ to $f' : X' \hookrightarrow Y'$ is a map $\alpha_2 : Y \to Y'$ for which there exists a unique map $\alpha_1 : X \to X'$ such that $\alpha_2 \circ f = f' \circ \alpha_1$. The map $U : Sub(\mathcal{B}) \to \mathcal{B}$ sending $f : X \hookrightarrow Y$ to Y extends to a functor. If $\mathcal{B}$ has pullbacks then U is a fibration since the pullback of a monomorphism is a monomorphism. In this case, U is called the *subobject fibration over* $\mathcal{B}$.

We also need the notion of an opfibration. Abstractly, $U : \mathcal{E} \to \mathcal{B}$ is an opfibration iff $U : \mathcal{E}^{op} \to \mathcal{B}^{op}$ is a fibration. More concretely, U is an opfibration if for every object P of $\mathcal{E}$ and every morphism $f : UP \to Y$ in $\mathcal{B}$ there is an *opcartesian morphism* $f^P_{\S} : P \to \Sigma_f P$ in $\mathcal{E}$ over f. Moreover, if $f : X \to Y$ is a morphism in $\mathcal{B}$, then the function mapping each object P of $\mathcal{E}_X$ to $\Sigma_f P$ extends to a functor $\Sigma_f : \mathcal{E}_X \to \mathcal{E}_Y$ which we call the *opreindexing functor*. A functor is a *bifibration* if it is both a fibration and an opfibration. The families and codomain fibrations are examples of bifibrations. More generally, a fibration is a bifibration iff, for every morphism $f : X \to Y$ in $\mathcal{B}$, f^* is left adjoint to Σ_f.

We can now give the key definitions and results for our fibrational approach to induction. If $U : \mathcal{E} \to \mathcal{B}$ is a fibration and $F : \mathcal{B} \to \mathcal{B}$ is a functor, then a *lifting* of F with respect to U is a functor $\hat{F} : \mathcal{E} \to \mathcal{E}$ such that $U\hat{F} = FU$. We say that U *has fibred terminal objects* if each fibre has a terminal object and reindexing functors preserve them. In this case, the functor $\top : \mathcal{B} \to \mathcal{E}$ mapping each object to the terminal object of the fibre over it is called the *truth functor* for U. A lifting $\hat{F}$ of F is called *truth-preserving* if $\top F = \hat{F}\top$.

Example 5. A truth-preserving lifting $F^{\to}$ of F with respect to the codomain fibration *cod* is given by the action of F on morphisms.

A *comprehension category with unit* (or *CCU*, for short) is a fibration $U : \mathcal{E} \to \mathcal{B}$ that has fibred terminal objects and is such that the terminal object functor $\top$ has a right adjoint $\{-\}$. In this case, $\{-\}$ is called the *comprehension functor* for U. If ϵ is the counit of the adjunction $\top \dashv \{-\}$, then defining $\pi_P = U\epsilon_P$ gives a *projection* natural transformation from $\{P\}$ to UP. Truth-preserving liftings with respect to CCUs are used in [9] to give induction rules. The key result is:

Theorem 3. *Let $U : \mathcal{E} \to \mathcal{B}$ be a CCU and $F : \mathcal{B} \to \mathcal{B}$ be a functor such that μF exists, and let $\hat{F}$ be a truth-preserving lifting of F. Then for every object P of $\mathcal{E}$ and every algebra $\alpha : \hat{F}P \to P$, there is a unique morphism $ind_F \alpha : \mu F \to \{P\}$ such that $\pi_P \circ ind_F\, \alpha = fold(U\alpha)$.*

The proof consists of constructing a right adjoint $\langle - \rangle : \mathsf{Alg}_{\hat{F}} \to \mathsf{Alg}_F$ mapping $\hat{F}$-algebras with carrier P to F-algebras with carrier $\{P\}$. Given an $\hat{F}$-algebra $\alpha : \hat{F}P \to P$, we can define $ind_F\, \alpha$ to be $fold\, \langle \alpha \rangle : \mu F \to \{P\}$. For second part of the theorem, first note that π_P is an F-algebra morphism from $\langle \alpha \rangle$ to $U\alpha$. Then, by the uniqueness of *fold*s, we have that $\pi_P \circ ind_F\, \alpha = \pi_P \circ fold \langle \alpha \rangle = fold(U\alpha)$.

Fibrations thus provide just the right structure for defining induction rules for inductive data types. Although the truth-preserving liftings are given in [9] only for polynomial functors, this restriction was removed in [7,8], which showed that in a *Lawvere fibration* — i.e., a CCU that is also a bifibration — *every* functor

has a truth-preserving lifting. Indeed, observing that π extends to a functor $\pi : \mathcal{E} \to \mathcal{B}^{\to}$ with left adjoint $I : \mathcal{B}^{\to} \to \mathcal{E}$ defined by $I\,(f : X \to Y) = \Sigma_f\,(\top X)$, we have that for any functor F, $\hat{F} = IF^{\to}\pi : \mathcal{E} \to \mathcal{E}$ is a truth-preserving lifting with respect to U, where $F^{\to}$ is the lifting from Example 5. If μF exists, then Theorem 3 guarantees that it has an induction rule.

5 Effectful Induction

In the remainder of the paper we assume a Lawvere fibration $U : \mathcal{E} \to \mathcal{B}$, a functor $F : \mathcal{B} \to \mathcal{B}$, and a monad (T, η, μ) on $\mathcal{B}$. We further assume that $\mu(FT)$ exists. Our first effectful induction rule is obtained by recalling that $T(\mu(FT))$ is the initial TF-algebra and instantiating Theorem 3 for TF.

Theorem 4. *For every object P of $\mathcal{E}$ and algebra $\alpha : \widehat{(TF)}P \to P$ there is a unique morphism $ind_{TF}\,\alpha : T(\mu(FT)) \to \{P\}$ with $\pi_P \circ ind_{TF}\,\alpha = fold(U\alpha)$.*

Unfortunately, the induction rule in Theorem 4 is more complicated than we would like since the rule requires the user to supply a $\widehat{TF}$-algebra, and thus to deal with F and T at the same time, rather than separately as in Corollary 1. To produce a fibrational variant of Corollary 1, we therefore need to understand the relationship between $\widehat{TF}$ and $\hat{T}\hat{F}$. We turn to this now.

Lemma 2. *If F and G are functors on $\mathcal{B}$, and $\alpha : F \to G$ is a natural transformation, then there is a natural transformation $\hat{\alpha} : \hat{F} \to \hat{G}$.*

Proof. Since I and π are functors, we can define $\hat{\alpha} = I\alpha^{\to}\pi$, where $\alpha^{\to} : F^{\to} \to G^{\to}$ maps $f : X \to Y$ in $\mathcal{B}^{\to}$ to the naturality square for α at f. □

Theorem 5. *The lifting operation $\widehat{(-)}$ defines a lax monoidal functor mapping functors on $\mathcal{B}$ to functors on $\mathcal{E}$.*

Proof. That $\widehat{(-)}$ preserves identity and composition of natural transformations is verified by simple calculation, so it is indeed a functor. To see that this functor is lax monoidal, we need natural transformations from $\widehat{Id}$ to Id and from $\widehat{FG}$ to $\hat{F}\hat{G}$. We take the former to be the counit of the adjunction $I \dashv \pi$. For the latter, define $\sigma : \widehat{FG} \to \hat{F}\hat{G}$ — i.e., $\sigma : I(FG)^{\to}\pi \to IF^{\to}\pi IG^{\to}\pi$ — by $\sigma = IF^{\to}\eta G^{\to}\pi$, where η is the unit of the adjunction $I \dashv \pi$.

We will use the natural transformation $\sigma : \widehat{FG} \to \hat{F}\hat{G}$ from the proof of Theorem 5 in the proof of Theorem 6 below. Note that if σ were oplax rather than lax — i.e., if we had $\sigma : \hat{F}\hat{G} \to \widehat{FG}$ — then $\hat{T}$ would be a monad whenever T is. An induction rule for effectful data types that assumes $\hat{T}$ is a monad is discussed in Section 6. For now, we derive induction rules for effectful data types that hold even when $\hat{T}$ is not a monad. The first is a fibrational variant of Corollary 1:

Theorem 6. *Let P be an object of $\mathcal{E}$, and let $f : \widehat{F}P \to P$ and $g : \widehat{T}P \to P$ be morphisms of $\mathcal{E}$. Then there is a unique morphism $h : T(\mu(FT)) \to \{P\}$ in $\mathcal{B}$ such that $\pi_P \circ h = fold(Ug \circ TUf)$. If P is over $T(\mu(FT))$, g is over μ, and f is over $\eta \circ in$, then $\pi_P \circ h = id$.*

Proof. From the algebra $\Phi(P,g,f) : \hat{T}\hat{F}P \to P$, where Φ is as in Lemma 1, we can construct the $\widehat{TF}$-algebra $\Phi(P,g,f) \circ \sigma_P$. By Theorem 4, there exists a morphism $h : T(\mu(FT)) \to \{P\}$ in $\mathcal{B}$ such that $\pi_P \circ h = fold(U(\Phi(P,g,f) \circ \sigma_P))$. If $f : X \to Y$ is an object of $\mathcal{B}^{\to}$, then η_f is a pair whose second component is id. The second component of $F^{\to}\eta_f$ is thus also id, so by the definition of I we have that $IF^{\to}\eta_f$ is vertical. Since $\sigma_P = IF^{\to}\eta_{G^{\to}\pi_P}$, σ_P is also vertical. Thus $fold(U(\Phi(P,g,f) \circ \sigma_P)) = fold(U(\Phi(P,g,f)))$, and by the definition of Φ and the fact that $\hat{T}$ is a lifting of T, we have that $\pi_P \circ h = fold(Ug \circ TUf)$ as desired. If g is over μ and f is over $\eta \circ in$, then $\pi_P \circ h = fold(Ug \circ TUf) = fold(\mu \circ T\eta \circ T(in)) = fold\,(T(in)) = fold\, in = id$. □

The condition $\pi_P \circ h = id$ ensures that h maps every element t of $T(\mu(FT))$ to a proof that Pt holds. We will make good use of the following generalisation of Prop. 2.13 in [5], which shows how to build new $\hat{T}$-algebras from old.

Lemma 3. *Let $k : TA \to A$ be an Eilenberg-Moore algebra for T.*

1. *Let δ be the natural transformation defined by $\delta_A : A \to A \times A$ and consider the equality predicate $Eq_A = \Sigma_\delta \top(A)$. Then $U(Eq_A) = A \times A$ and $\langle k \circ T\pi_1, k \circ T\pi_2\rangle : T(A \times A) \to A \times A$ is an Eilenberg-Moore algebra for T. If $\{Eq_A\} = A$ for every $A \in \mathcal{B}$, then there exists a morphism $h : \hat{T}\, Eq_A \to Eq_A$ such that $Uh = \langle k \circ T\pi_1, k \circ T\pi_2\rangle$.*
2. *Let $h : \hat{T}P \to P$ be a $\hat{T}$-algebra such that $UP = A$ and $Uh = k$. Then for every Eilenberg-Moore algebra $k' : TB \to B$ for T and T-algebra morphism $f : B \to A$, there exists a morphism $h' : \hat{T}(f^*P) \to f^*P$ such that $Uh' = k'$.*
3. *Let I be a set and suppose $\mathcal{E}$ has I-indexed products in its fibres. Then for any I-indexed family $(P_i, h_i : \hat{T}P_i \to P_i)$ of $\hat{T}$-algebras with $UP_i = A$ and $Uh_i = k$ for all i, there is a morphism $h : \hat{T}(\Pi_{i\in I}P_i) \to \Pi_{i\in I}P_i$ with $Uh = k$.*

Proof. The first part follows from a lemma in [10], and the third part follows from the universal property of products. For the second part, note that because h is over k, there is a vertical morphism $v : \hat{T}P \to k^*P$. We construct $h' : \hat{T}(f^*P) \to P$ by first noting that, since f is a T-Eilenberg-Moore algebra morphism, we have $(Tf)^*(k^*P) = k'^*(f^*P)$. We can then take h' to be the composition of $j : \hat{T}(f^*P) \to (Tf)^*(\hat{T}P)$ and $(Tf)^*v : (Tf)^*(\hat{T}P) \to (Tf)^*(k^*P)$ and $k'^{\S}_{f^*P} : k'^*(f^*P) \to f^*P$. Here, j is constructed by first observing that $\hat{T}(f^*P) = \Sigma_{T\pi_{f^*P}}\top(T\{f^*P\})$ by definition of $\hat{T}$, and that this is $\Sigma_{T\pi_{f^*P}}(T\{f^{\S}_P\})^*\top(T\{P\})$ because reindexing preserves terminal objects. We then construct a morphism from $\Sigma_{T\pi_{f^*P}}(T\{f^{\S}_P\})^*\top(T\{P\})$ to $(Tf)^*\Sigma_{T\pi_P}\top(T\{P\})$ using a morphism induced by a natural transformation from $\Sigma_{T\pi_{f^*P}}(T\{f^{\S}_P\})^*$ to $(Tf)^*\Sigma_{T\pi_P}$ that is itself constructed using i) the fact that f is a T-algebra morphism, ii) the unit of the adjunction $\Sigma_{T\pi_P} \dashv (T\pi_P)^*$, and iii) the counit of the adjunction $\Sigma_{T\pi_{f^*P}} \dashv (T\pi_{f^*P})^*$. Noting that $(Tf)^*\Sigma_{T\pi_P}\top(T\{P\}) = (Tf)^*(\hat{T}P)$ by the definition of $\hat{T}$ completes the proof. □

Example 6. We can now prove that `appIO` is associative. We use the families fibration, and so exploit the generality of our framework over that of [5] by working

with a base category other than CPO and a notion of predicate other than that given by subobjects. The predicate P : `IOList a` $\to \mathsf{Set}$ sends `xs` to 1 if, for all `ys zs: IOList a`, `appIO xs (appIO ys zs) = appIO (appIO xs ys) zs` holds and to $\emptyset$ otherwise. Recalling that `IOList a` is $T(\mu(FT))$ for F and T as in Example 1, we can show that P is the carrier of a $\hat{T}$-algebra by observing that i) the equality predicate on `IOList a` is the carrier of a T-algebra over $\langle \mu \circ T\pi_1, \mu \circ T\pi_2 \rangle$ by the first part of Lemma 3; ii) for all `ys zs : IOList a`, the predicate `appIO xs (appIO ys zs) = appIO (appIO xs ys) zs` on `IOList a` $\times$ `IOList a` is the carrier of a $\hat{T}$-algebra over μ by the second part of Lemma 3; and iii) P is thus the carrier of a $\hat{T}$-algebra over μ by the third part of Lemma 3. In addition, P is the carrier of an $\hat{F}$-algebra: indeed by direct calculation we have that the predicate $\hat{F}P : FX \to \mathsf{Set}$ sends $inl\, *$ to 1 and sends $inr(x, xs)$ to $P\, xs$. An $\hat{F}$-algebra with carrier P over $\eta \circ in$ is therefore given by an element of P(`return IONil`) and, for every `x:a` and `xs : IOList a`, a function P `xs` $\to P$ (`return (IOCons x xs)`). Both can be computed directly using the definition of `appIO`. By Theorem 6, we thus have that P holds for all elements of `IOList a`.

Example 7. We can exploit our ability to move beyond the effectful strictly positive data types treated by Filinski and Støvring to reason about *indexed effectful data types*. We work in the subobject fibration over $\mathsf{Set}^{\mathbb{N}}$ (see Example 4). For any set A and monad T on $\mathsf{Set}^{\mathbb{N}}$, consider the $\mathbb{N}$-indexed data type of effectful perfect trees with data from A and effects from T given by $T(\mu(F_A T)) : \mathsf{Set}^{\mathbb{N}}$, where, $F_A(X : \mathsf{Set}^{\mathbb{N}}) = \lambda n.\{* \mid n = 0\} + \{(a, x_1, x_2) \mid \exists n'.\ a \in A,\ x_1 \in Xn',\ x_2 \in Xn', n = n' + 1\}$. By Theorem 2, if $f : A \to B$ then we can define a morphism $map(f, -) : T(\mu(F_A T)) \to T(\mu(F_B T))$ in $\mathsf{Set}^{\mathbb{N}}$ by giving an F_A-and-T-Eilenberg-Moore algebra with carrier $T(\mu(F_B T))$. The F_A-algebra sends, when $n = 0$, $inl\, *$ to $\eta(in(inl\, *))$, and, when $n = n' + 1$, $inr(a, x_1, x_2)$ to $\eta(in(inr(fa, x_1, x_2)))$. The Eilenberg-Moore algebra for T is just the multiplication μ of T. Now define the subobject $i : P \hookrightarrow T(\mu(F_A T))$ in $Sub(\mathsf{Set}^{\mathbb{N}})$ by $Pn = \{t : T(\mu(F_A T))n \mid \forall f, g.\ map(f, map(g, t)) = map(f \circ g, t)\}$. To show that $P = T(\mu(F_A T))$, and hence that map preserves composition, we apply Theorem 6. As in Example 6, we use Lemma 3 to give a $\hat{T}$-algebra on i over μ; this uses the fact that $map(f, -)$ is a T-Eilenberg-Moore algebra morphism by construction. The existence of a $\hat{F}$-algebra on i over $\eta \circ in$ follows by direct calculation. By Theorem 6 there is a morphism $h : T(\mu(F_A T)) \to \{P\}$ such that $\pi_P \circ h = id$. But since $\{P\} = P$ in $Sub(\mathsf{Set}^{\mathbb{N}})$, we have that i and h together give $P = T(\mu(F_A T))$.

Should we be satisfied with the effectful induction rule in Theorem 6? It is not as expressive as that in Theorem 4 since not all $\widehat{TF}$-algebras arise from $\hat{T}$-algebras and $\hat{F}$-algebras individually, but it is easier to use since we can check whether or not predicates have $\hat{T}$-algebra structures without considering the functor $\hat{F}$ at all and vice-versa. However, neither rule ensures that h respects the structure of the monad, i.e., is an F-and-T-Eilenberg-Moore algebra morphism. As we now see, this is the case if $\langle g \rangle$ is an Eilenberg-Moore algebra for T.

Theorem 7. *Let $f : \widehat{F}P \to P$ and $g : \widehat{T}P \to P$ be morphisms of $\mathcal{E}$ such that $\langle g \rangle$ is an Eilenberg-Moore algebra for T. Then there is a unique $h : T(\mu(FT)) \to \{P\}$ in $\mathcal{B}$ that is an F-and-T-Eilenberg-Moore algebra morphism. Further, $\pi_p \circ h = fold\,(U f \circ FU g)$. If g is over μ and f is over $\eta \circ in$, then $\pi_p \circ h = id$.*

Proof. Since $(\{P\}, \langle f \rangle, \langle g \rangle)$ is an F-and-T-Eilenberg-Moore algebra there is a unique F-and-T-Eilenberg-Moore algebra morphism from the initial such algebra to it. Since $T(\mu(FT))$ is the carrier of the initial F-and-T-Eilenberg-Moore algebra, this gives the required morphism $h : T(\mu(FT)) \to \{P\}$. Indeed, h is the morphism guaranteed by Theorem 6, and so $\pi_p \circ h = id$ as desired. □

As expected, the effectful world contains the pure world. For example, if (T, η, μ) is a monad, then we can think of η as a family of functions $\eta_X : X \to TX$ mapping values of type X to the pure computations that just return those values. Similarly, an effectful data structure $T(\mu(FT))$ contains the pure data structure μF: indeed, the natural transformation $\eta F : F \to TF$ and the functoriality of the fixed point operator μ together give the inclusion $\mu(\eta F) : \mu F \to T(\mu(FT))$. Moreover, if P is a property over $T(\mu(FT))$, then $(\mu(\eta F))^* P$ is a property over μF. Thus given $f : \hat{F}P \to P$ and $g : \hat{T}P \to P$ such that $\langle g \rangle$ is an Eilenberg-Moore algebra for T, we may ask about the relationship between proofs of $(\mu(\eta F))^* P$ for μF obtained from f by Theorem 3 and proofs of P for $T(\mu(FT))$ obtained from f and g by Theorem 7. It is not hard to see that induction for $T(\mu(FT))$ specialises to induction for μF when $T(\mu(FT))$ is pure.

Finally, note that in a fibred preorder, for any $\hat{T}$-algebra g, $\langle g \rangle$ is always an Eilenberg-Moore algebra for T. This is the case for the subobject fibration implicitly used by Filinski and Støvring since, there, admissible predicates are Eilenberg-Moore algebras over the multiplication μ of T. However, in the non-fibred preorder case, there are a variety of different induction rules which are possible. This reflects a trade-off: the more we assume, the better behaved our induction proof is! Our default preference is for structure and we believe that Theorem 7 provides the best rule. However, if we cannot establish the stronger premises so as to obtain the better behaved induction rules, it is comforting to know that the induction rules of Theorems 4 and 6 are still available.

6 A More Logical Treatment of Effectful Induction

The treatment we have given above of induction for effectful data types addresses many of our practical concerns. It is derived from the general theory of fibrational induction, it is axiomatic in terms of the functors, monads, and fibrations involved, and Theorems 6 and 7 allow us to separate the proof obligations in an induction proof into those pertaining only to the monad in question and those pertaining only to the functor in question. There is, however, one feature of Theorem 7 that is less than optimal. Underlying the fibrational methodology is the separation between logical structure in the total category of the fibration and type-theoretic structure in the base category of the fibration. Theorem 7 can thus be seen as converting logical structure in the form of $\hat{F}$-algebras and $\hat{T}$-algebras

into type-theoretic structure in the form of F-and-T-Eilenberg-Moore algebras. This is, of course, completely valid, especially in light of the *propositions-as-types* interpretation, but this section shows there is a different approach which reasons solely in the total category of the fibration and is thus purely logical. The key idea is to deploy Theorem 2 in the total category of the fibration, and work directly with $\hat{F}$ and $\hat{T}$-algebras on P rather than converting them to F-and-T-Eilenberg-Moore algebras on $\{P\}$. The major stumbling block to doing this is that, in general, $\hat{T}$ is not a monad. In this section we investigate effectful induction in the case when $\hat{T}$ is a monad. The condition we use to ensure this is that the Lawvere fibration in which we work has *very strong sums*.

Definition 3 *A Lawvere fibration $U : \mathcal{E} \to \mathcal{B}$ is said to have very strong sums if for all $f : X \to Y$ in $\mathcal{B}$ and $P \in \mathcal{E}_X$, $\{f^{\S}_P\} : \{P\} \to \{\Sigma_f P\}$ is an isomorphism.*

The following important property of very strong sums is from [1]: If $U : \mathcal{E} \to \mathcal{B}$ is a Lawvere fibration with very strong sums, $F : \mathcal{B} \to \mathcal{B}$ is a functor, $f : X \to Y$ is a morphism, and $P \in \mathcal{E}_X$, then $\hat{F}(\Sigma_f P) = \Sigma_{Ff} \hat{F} P$. Using this, we can prove that in a Lawvere fibration with very strong sums lifting is actually a strong monoidal functor, i.e., that lifting preserves functor composition.

Lemma 4. *Let $U : \mathcal{E} \to \mathcal{B}$ be a Lawvere fibration with very strong sums. If $F, G : \mathcal{B} \to \mathcal{B}$ are functors, then $\widehat{FG} = \hat{F}\hat{G}$. If T is a monad, then so is $\hat{T}$.*

Proof. For the first part of the lemma, note that $(\widehat{FG})P = \Sigma_{FG\pi_P} K_1 FG\{P\} = \Sigma_{FG\pi_P} \hat{F} K_1 G\{P\} = \hat{F}(\Sigma_{G\pi_P} K_1 G\{P\}) = \hat{F}\hat{G}P$. Here, the first equality is by the definition of $\widehat{FG}$, the second holds because $\hat{F}$ is truth preserving, the third is by the aforementioned property from [1], and the last is by definition of $\hat{G}$. The essence of the proof of the second part is the observations that monads are just monoids in the monoidal category of endofunctors and strong monoidal functors map monoids to monoids. □

Using Lemma 4 we can derive induction rules allowing us to work as much as possible in the total category of a Lawvere fibration with very strong sums. To do this, let (P, f, g) be an $\hat{F}$-and-$\hat{T}$-Eilenberg-Moore algebra. By Theorem 2, there is a unique morphism from the initial $\hat{T}\hat{F}$-algebra to (P, f, g). But since $\hat{T}\hat{F} = \widehat{TF}$ and $\mu(\widehat{TF}) = \top(\mu(TF))$ (see Corollary 4.10 of [8]), there is a morphism from $\top(T(\mu(FT)))$ to P, and thus one from $T(\mu(FT))$ to $\{P\}$ as desired.

The families and codomain fibrations both have very strong sums.

7 Conclusions, Related Work, and Future Work

We have investigated the interaction between induction and effects. We formalised the former using the recently developed fibrational interpretation of induction because it is axiomatic in the category interpreting types and programs, the functor representing the data type in question, and the category interpreting predicates. We formalised effects using monads because they are both simple to understand and widely used. We have shown, perhaps surprisingly, that several

induction rules can be derived for effectful data types. These rules assume progressively more structure in their hypotheses but deliver progressively stronger inductive proofs. Ultimately, we hope this research will lay the foundation for a reasoning module for effectful data types in a proof system such as Coq.

The combination of monadic effects and inductive data types has previously been studied by Fokkinga [4] and Pardo [14]. They use distributive laws $\lambda : FT \to TF$ relating functors describing data types to monads modelling effects. Given a distributive law, it can be shown that μF is the carrier of an initial algebra in the Kleisli category of T. From this, a theory of effectful structural recursion over *pure* data is derived. By contrast, in this paper we have explored computation and reasoning with *effectful* data, where data and effects are interleaved.

Lehman and Smyth [12] give a generic induction rule for (pure) inductive data types in the case when predicates are taken to be subobjects in a category. Crole and Pitts [3] use this rule to give a fixpoint induction rule for effectful computations, generalising the usual notion of Scott induction. Filinski and Støvring's induction principle, which we have generalised in this paper, extends these rules to handle the interleaving of data and effects.

References

1. R. Atkey, N. Ghani, and P. Johann. When is a Type Refinement an Inductive Type? *Proc., Foundations of Software Science and Computation Structures*, pp. 72–87, 2011.
2. R. Burstall. Proving Properties of Programs by Structural Induction. *Computer Journal* 12(1), pp. 41–48, 1969.
3. R. Crole and A. Pitts. New Foundations for Fixpoint Computations: FIX-Hyperdoctrines and the FIX-Logic. *Information and Computation* 98(2), pp. 171–210, 1992.
4. M. Fokkinga. Monadic Maps and Folds for Arbitrary Datatypes. Technical Report, University of Twente, 1994.
5. A. Filinski and K. Støvring. Inductive Reasoning About Effectful Data Types. *Proc., International Conference on Functional Programming*, pp. 97–110, 2007.
6. A. Gill and G. Hutton. The worker/wrapper Transformation. *Journal of Functional Programming* 19(2), pp. 227–251, 2009.
7. N. Ghani, P. Johann, and C. Fumex. Fibrational Induction Rules for Initial Algebras. *Proc., Computer Science Logic*, pp. 336–350, 2010.
8. N. Ghani, P. Johann, and C. Fumex. Generic Fibrational Induction. Submitted, 2011.
9. C. Hermida and B. Jacobs. Structural Induction and Coinduction in a Fibrational Setting. *Information and Computation* 145, pp. 107–152, 1998.
10. B. Jacobs. *Categorical Logic and Type Theory*. Studies in Logic and the Foundations of Mathematics, Volume 141, Elsevier, 1999.
11. B. Jacobs. Comprehension Categories and the Semantics of Type Dependency. *Theoretical Computer Science* 107, pp. 169–207, 1993.
12. D. Lehmann and M. Smyth. Algebraic Specification of Data Types: A Synthetic Approach. *Theory of Computing Systems* 14(1), pp. 97–139, 1981.
13. E. Moggi. Computational Lambda-Calculus and Monads. *Proc., Logic in Computer Science*, pp. 14–23, 1989.
14. A. Pardo. Combining Datatypes and Effects. *Proc., Advanced Functional Programming*, LNCS 3622, pp.171–209, 2004.