Archived version from NCDOCKS Institutional Repository http://libres.uncg.edu/ir/asu/

Appalachian
STATE UNIVERSITY
BOONE, NORTH CAROLINA

Generic Fibrational Induction

By: Ghani, Neil; **Johann, Patricia;** and Fumex, Clement

Abstract

This paper provides an induction rule that can be used to prove properties of data structures whose types are inductive, i.e., are carriers of initial algebras of functors. Our results are semantic in nature and are inspired by Hermida and Jacobs' elegant alge-braic formulation of induction for polynomial data types. Our contribution is to derive, under slightly different assumptions, a sound induction rule that is generic over all inductive types, polynomial or not. Our induction rule is generic over the kinds of properties to be proved as well: like Hermida and Jacobs, we work in a general fibrational setting and so can accommodate very general notions of properties on inductive types rather than just those of a particular syntactic form. We establish the soundness of our generic induction rule by reducing induction to iteration. We then show how our generic induction rule can be instantiated to give induction rules for the data types of rose trees, finite hereditary sets, and hyperfunctions. The first of these lies outside the scope of Hermida and Jacobs' work because it is not polynomial, and as far as we are aware, no induction rules have been known to exist for the second and third in a general fibrational framework. Our instantiation for hyperfunctions underscores the value of working in the general fibrational setting since this data type cannot be interpreted as a set.

Ghani, Neil; **Johann, Patricia**; and Fumex, Clement (2012) "Generic Fibrational Induction". *Logical Methods in Computer Science* 8(2), 2012. Published: June 19, [DOI: 10.2168/LMCS-8(2:12)] Version of Record Available From www.arvix.org

GENERIC FIBRATIONAL INDUCTION

NEIL GHANI, PATRICIA JOHANN, AND CLÉMENT FUMEX

1. Introduction

Iteration operators provide a uniform way to express common and naturally occurring patterns of recursion over inductive data types. Expressing recursion via iteration operators makes code easier to read, write, and understand; facilitates code reuse; guarantees properties of programs such as totality and termination; and supports optimising program transformations such as *fold* fusion and short cut fusion. Categorically, iteration operators arise from initial algebra semantics of data types, in which data types are regarded as carriers of initial algebras of functors. Lambek's Lemma ensures that the carrier of the initial algebra of F is its least fixed point μF, and initiality ensures that, given any F-algebra $h : FA \to A$, there is a unique F-algebra homomorphism, denoted $fold\, h$, from the initial algebra $in : F(\mu F) \to \mu F$ to that algebra. For each functor F, the map *fold* : $(FA \to A) \to \mu F \to A$ is the iteration operator for the data type μF. Initial algebra semantics thus provides a well-developed theory of iteration which is...

- *...principled*, in that it is derived solely from the initial algebra semantics of data types. This is important because it helps ensure that programs have rigorous mathematical foundations that can be used to ascertain their meaning and correctness.

- *...expressive*, and so is applicable to all inductive types — i.e., to *every* type which is the carrier of an initial algebra of a functor — rather than just to syntactically defined classes of data types such as polynomial data types.

- *...correct*, and so is valid in any model — set-theoretic, domain-theoretic, realisability, etc. — in which data types are interpreted as carriers of initial algebras.

Because induction and iteration are closely linked — induction is often used to prove properties of functions defined by iteration, and the correctness of induction rules is often established by reducing it to that of iteration — we may reasonably expect that initial algebra semantics can be used to derive a principled, expressive, and correct theory of induction for data types as well. In most treatments of induction, given a functor F together with a property P to be proved about data of type μF, the premises of the induction rule for μF constitute an F-algebra with carrier $\Sigma x : \mu F.\, Px$. The conclusion of the rule is obtained by supplying such an F-algebra as input to the iteration operator for μF. This yields a function from μF to $\Sigma x : \mu F.\, Px$ from which a function of type $\forall x : \mu F.\, Px$ can be obtained. It has not, however, been possible to characterise F-algebras with carrier $\Sigma x : \mu F.\, Px$ without additional assumptions on F. Induction rules are thus typically derived under the assumption that the functors involved have a certain structure, e.g., that they are polynomial. Moreover, taking the carriers of the algebras to be Σ-types assumes that properties are represented as type-valued functions. So while induction rules derived as described above are both principled and correct, their expressiveness is limited along two dimensions: with respect to the data types for which they can be derived and the nature of the properties they can verify.

A more expressive, yet still principled and correct, approach to induction is given by Hermida and Jacobs [10]. They show how to lift each functor F on a base category of types to a functor $\hat{F}$ on a category of properties over those types, and take the premises of the induction rule for the type μF to be an $\hat{F}$-algebra. Hermida and Jacobs work in a fibrational setting and the notion of property they consider is, accordingly, very general. Indeed, they accommodate any notion of property that can be suitably fibred over the category of types, and so overcome one of the two limitations mentioned above. On the other hand, their approach gives sound induction rules only for polynomial data types, so the limitation on the class of data types treated remains in their work.

This paper shows how to remove the restriction on the class of data types treated. *Our main result is a derivation of a sound generic induction rule that can be instantiated to* every *inductive type*, regardless of whether it is polynomial or not. We think this is important because it provides a counterpart for induction to the existence of an iteration operator for every inductive type. We take Hermida and Jacobs' approach as our point of departure and show that, under slightly different assumptions on the fibration involved, we can lift *any* functor on the base category of a fibration to a functor on the total category of the fibration. The lifting we define forms the basis of our generic induction rule.

The derivation of a generic, sound induction rule covering all inductive types is clearly an important theoretical result, but it also has practical consequences:

- We show in Example 2 how our generic induction rule can be instantiated to the families fibration over Set (the fibration most often implicitly used by type theorists and those constructing inductive proofs with theorem provers) to derive the induction rule for rose trees that one would intuitively expect. The data type of rose trees lies outside the scope of Hermida and Jacobs' results because it is not polynomial. On the other hand, an induction rule for rose trees is available in the proof assistant Coq, although it is neither the one we intuitively expect nor expressive enough to prove properties that ought to be amenable to inductive proof. Indeed, if we define rose trees in Coq by

  ```
  Node : list rose -> rose
  ```

  then Coq generates the following induction rule

  ```
  rose_ind : forall P : rose -> Prop,
                 (forall l : list rose, P (Node l)) ->
                 forall r : rose, P r
  ```

  But to prove a property of a rose tree `Node l`, we must prove that property assuming only that `l` is a list of rose trees, and without recourse to any induction hypothesis. There is, of course, a presentation of rose trees by mutual recursion as well, but this doesn't give the expected induction rule in Coq either. Intuitively, what we expect is an induction rule whose premise is

  ```
  forall [r_0, ..., r_n] : list rose,
       P(r_0) -> ... -> P(r_n) -> P(Node [r_0, ..., r_n])
  ```

  The rule we derive for rose trees is indeed the expected one, which suggests that our derivation may enable automatic generation of more useful induction rules in Coq, rather than requiring the user to hand code them as is currently necessary.

- We further show in Example 3 how our generic induction rule can be instantiated, again to the families fibration over Set, to derive a rule for the data type of finite hereditary sets. This data type is defined in terms of quotients and so lies outside most current theories of induction.

- Finally, we show in Example 7 how our generic induction rule can be instantiated to the subobject fibration over $\omega CPO_\perp$ to derive a rule for the data type of hyperfunctions. Because this data type cannot be interpreted as a set, a fibration other than the families fibration over Set is required; in this case, use of the subobject fibration allows us to derive an induction rule for admissible subsets of hyperfunctions. The ability to treat the data type of hyperfunctions thus underscores the importance of developing our results in the general fibrational framework. Moreover, the functor underlying the data type of hyperfunctions is not strictly positive [7], so the ability to treat this data type also underscores the advantage of being able to handle a very general class of functors going beyond simply polynomial functors. As far as we know, induction rules for finite hereditary sets and hyperfunctions have not previously existed in the general fibrational framework.

Although our theory of induction is applicable to *all* inductive functors — i.e., to *all* functors having initial algebras, including those giving rise to nested types [15], GADTs [21], indexed containers [1], dependent types [19], and inductive recursive types [6] — our examples show that working in the general fibrational setting is beneficial even if we restrict our attention to strictly positive data types. We do, however, offer some preliminary thoughts in Section 5 on the potentially delicate issue of instantiating our general theory with fibrations appropriate for deriving induction rules for specific classes of higher-order functors of interest. It is also worth noting that the specialisations of our generic induction rule to polynomial functors in the families fibration over Set coincide exactly with the induction rules of Hermida and Jacobs. But the structure we require of fibrations generally is slightly different from that required by Hermida and Jacobs, so while our theory is in essence a generalisation of theirs, the two are, strictly speaking, incomparable. The structure we require of our fibrations is, nevertheless, certainly present in all standard fibrational models of type theory (see Section 4). Like Hermida and Jacobs, we prove our generic induction rule correct by reducing induction to iteration. A more detailed discussion of when our induction rules coincide with those of Hermida and Jacobs is given in Section 4.

We take a purely categorical approach to induction in this paper, and derive our generic induction rule from only the initial algebra semantics of data types. As a result, our work is inherently extensional. Although translating our constructions into intensional settings may therefore require additional effort, we expect the guidance offered by the categorical viewpoint to support the derivation of induction rules for functors that are not treatable at present. Since we do not use any form of impredicativity in our constructions, and instead use only the weaker assumption that initial algebras exist, this guidance will be widely applicable.

The remainder of this paper is structured as follows. To make our results as accessible as possible, we illustrate them in Section 2 with a categorical derivation of the familiar induction rule for the natural numbers. In Section 3 we derive an induction rule for the special case of the families fibration over Set. We also show how this rule can be instantiated to derive the one from Section 2, and the ones for rose trees and finite hereditary sets mentioned above. Then, in Section 4 we present our generic fibrational induction rule, establish a number of results about it, and illustrate it with the aforementioned application to hyperfunctions. The approach taken in this section is completely different from the corresponding one in the conference version of the paper [9], and allows us to improve upon and extend our previous results. Section 5 concludes, discusses possible instantiations of our generic induction rule for higher-order functors, and offers some additional directions for future research.

When convenient, we identify isomorphic objects of a category and write $=$ rather than $\simeq$. We write 1 for the canonical singleton set and denote its single element by $\cdot$. In Sections 2 and 3 we assume that types are interpreted as objects in Set, so that 1 also denotes the unit type in those sections. We write *id* for identity morphisms in a category and *Id* for the identity functor on a category.

2. A Familiar Induction Rule

Consider the inductive data type Nat, which defines the natural numbers and can be specified in a programming language with Haskell-like syntax by

$$data\ Nat \ = \ Zero \mid Succ\, Nat$$

The observation that Nat is the least fixed point of the functor N on Set — i.e., on the category of sets and functions — defined by $NX \ = \ 1 + X$ can be used to define the following iteration operator:

$$\begin{array}{lcl} foldNat & = & X \to (X \to X) \to Nat \to X \\ foldNat\, z\, s\, Zero & = & z \\ foldNat\, z\, s\, (Succ\, n) & = & s\, (foldNat\, z\, s\, n) \end{array}$$

The iteration operator *foldNat* provides a uniform means of expressing common and naturally occurring patterns of recursion over the natural numbers.

Categorically, iteration operators such as *foldNat* arise from the initial algebra semantics of data types, in which every data type is regarded as the carrier of the initial algebra of a functor F. If $\mathcal{B}$ is a category and F is a functor on $\mathcal{B}$, then an *F-algebra* is a morphism $h : FX \to X$ for some object X of $\mathcal{B}$. We call X the *carrier* of h. For any functor F, the collection of F-algebras itself forms a category Alg_F which we call the *category of F-algebras*. In Alg_F, an *F-algebra morphism* between F-algebras $h : FX \to X$ and $g : FY \to Y$ is a map $f : X \to Y$ such that the following diagram commutes:

$$\begin{tikzcd} FA \arrow[r, "Ff"] \arrow[d, "h"'] & FB \arrow[d, "g"] \\ A \arrow[r, "f"] & B \end{tikzcd}$$

When it exists, the initial F-algebra $in : F(\mu F) \to \mu F$ is unique up to isomorphism and has the least fixed point μF of F as its carrier. Initiality ensures that there is a unique F-algebra morphism $fold\ h : \mu F \to X$ from in to any F-algebra $h : FX \to X$. This gives rise to the following iteration operator $fold$ for the inductive type μF:

$$\begin{array}{lcl} fold & : & (FX \to X) \to \mu F \to X \\ fold\, h\, (in\, t) & = & h\, (F\, (fold\, h)\, t) \end{array}$$

Since *fold* is derived from initial algebra semantics it is principled and correct. It is also expressive, since it can be defined for *every* inductive type. In fact, *fold* is a single iteration operator parameterised over inductive types rather than a family of iteration operators, one for each such type, and the iteration operator *foldNat* above is the instantiation to Nat of the generic iteration operator *fold*.

The iteration operator *foldNat* can be used to derive the standard induction rule for Nat which coincides with the standard induction rule for natural numbers, i.e., with the familiar principle of mathematical induction. This rule says that if a property P holds for 0, and if P holds for $n+1$ whenever it holds for a natural number n, then P holds for all natural numbers. Representing each property of natural numbers as a predicate $P : Nat \to \mathsf{Set}$ mapping each $n : Nat$ to the set of proofs that P holds for n, we wish to represent this rule at the object level as a function *indNat* with type

$$\forall (P : Nat \to \mathsf{Set}). \quad P\, Zero \to (\forall n : Nat.\, P\, n \to P\, (Succ\, n)) \to (\forall n : Nat.\, P\, n)$$

Code fragments such as the above, which involve quantification over sets, properties, or functors, are to be treated as "categorically inspired" within this paper. This is because quantification over such higher-kinded objects cannot be interpreted in Set. In order to give a formal interpretation to code fragments like the one above, we would need to work in a category such as that of modest sets. While the ability to work with functors over categories other than Set is one of the motivations for working in the general fibrational setting of Section 4, formalising the semantics of such code fragments would obscure the central message of this paper. Our decision to treat such fragments as categorically inspired is justified in part by the fact that the use of category theory to suggest computational constructions has long been regarded as fruitful within the functional programming community (see, e.g., [2, 3, 18]).

A function *indNat* with the above type takes as input the property P to be proved, a proof ϕ that P holds for *Zero*, and a function ψ mapping each $n : \mathit{Nat}$ and each proof that P holds for n to a proof that P holds for $\mathit{Succ}\, n$, and returns a function mapping each $n : \mathit{Nat}$ to a proof that P holds for n, i.e., to an element of $P\, n$. We can write *indNat* in terms of *foldNat* — and thus reduce induction for *Nat* to iteration for *Nat* — as follows. First note that *indNat* cannot be obtained by instantiating the type X in the type of *foldNat* to a type of the form Pn for a specific n because *indNat* returns elements of the types $P\, n$ for different values n and these types are, in general, distinct from one another. We therefore need a type containing all of the elements of $P\, n$ for every n. Such a type can informally be thought of as the union over n of Pn, and is formally given by the dependent type $\Sigma n : \mathit{Nat}.\, P\, n$ comprising pairs (n, p) where $n : \mathit{Nat}$ and $p : P\, n$.

The standard approach to defining *indNat* is thus to apply *foldNat* to an N-algebra with carrier $\Sigma n : \mathit{Nat}.\, P\, n$. Such an algebra has components $\alpha : \Sigma n : \mathit{Nat}.\, P\, n$ and $\beta : \Sigma n : \mathit{Nat}.\, P\, n \;\to\; \Sigma n : \mathit{Nat}.\, P\, n$. Given $\phi : P\, \mathit{Zero}$ and $\psi : \forall n.\, P\, n \to P\, (\mathit{Succ}\, n)$, we choose $\alpha \;=\; (\mathit{Zero}, \phi)$ and $\beta\, (n, p) \;=\; (\mathit{Succ}\, n, \psi\, n\, p)$ and note that $\mathit{foldNat}\, \alpha\, \beta : \mathit{Nat} \to \Sigma n : \mathit{Nat}.\, P\, n$. We tentatively take $\mathit{indNat}\ P\ \phi\ \psi\ n$ to be p, where $\mathit{foldNat}\, \alpha\, \beta\, n \;=\; (m, p)$. But in order to know that p actually gives a proof for n itself, we must show that $m = n$. Fortunately, this follows easily from the uniqueness of $\mathit{foldNat}\, \alpha\, \beta$. Indeed, we have that

$$\begin{tikzcd}
1 + \mathit{Nat} \arrow[r] \arrow[d, "in"'] & 1 + \Sigma n : \mathit{Nat}.\, P\, n \arrow[r] \arrow[d, "{[\alpha,\beta]}"'] & 1 + \mathit{Nat} \arrow[d, "in"'] \\
\mathit{Nat} \arrow[r, "\mathit{foldNat}\, \alpha\, \beta"] & \Sigma n : \mathit{Nat}.\, P\, n \arrow[r, "\lambda(n,p).\, n"] & \mathit{Nat}
\end{tikzcd}$$

commutes and, by initiality of *in*, that $(\lambda(n, p).\, n) \circ (\mathit{foldNat}\, \alpha\, \beta)$ is the identity map. Thus

$$n = (\lambda(n, p).\, n)(\mathit{foldNat}\, \alpha\, \beta\, n) = (\lambda(n, p).\, n)(m, p) = m$$

Letting π'_P be the second projection on dependent pairs involving the predicate P, the induction rule for *Nat* is thus

$$\begin{array}{lcl}
\mathit{indNat} & : & \forall (P : \mathit{Nat} \to \mathsf{Set}).\, P\, \mathit{Zero} \to (\forall n : \mathit{Nat}.\, P\, n \to P\, (\mathit{Succ}\, n)) \\
 & & \quad \to (\forall n : \mathit{Nat}.\, P\, n) \\
\mathit{indNat}\ P\ \phi\ \psi & = & \pi'_P \circ (\mathit{foldNat}\ (\mathit{Zero}, \phi)\ (\lambda(n, p).\, (\mathit{Succ}\, n, \psi\, n\, p)))
\end{array}$$

As expected, this induction rule states that, for every property P, to construct a proof that P holds for every $n : \mathit{Nat}$, it suffices to provide a proof that P holds for *Zero*, and to show that, for any $n : \mathit{Nat}$, if there is a proof that P holds for n, then there is also a proof that P holds for $\mathit{Succ}\, n$.

The use of dependent types is fundamental to this formalization of the induction rule for *Nat*, but this is only possible because properties to be proved are taken to be set-valued functions. The remainder of this paper uses fibrations to generalise the above treatment of induction to arbitrary inductive functors and arbitrary properties which are suitably fibred over the category whose objects interpret types. In the general fibrational setting, properties are given axiomatically via the fibrational structure rather than assumed to be (set-valued) functions.

3. Induction Rules for Predicates over Set

The main result of this paper is the derivation of a sound induction rule that is generic over all inductive types and which can be used to verify any notion of property that is fibred over the category whose objects interpret types. In this section we assume that types are modelled by sets, so the functors we consider are on Set and the properties we consider are functions mapping data to sets of proofs that these properties hold for them. We make these assumptions because it allows us to present our derivation in the simplest setting possible, and also because type theorists often model properties in exactly this way. This makes the present section more accessible and, since the general fibrational treatment of induction can be seen as a direct generalisation of the treatment presented here, Section 4 should also be more easily digestible once the derivation is understood in this special case. Although the derivation of this section can indeed be seen as the specialisation of that of Section 4 to the families fibration over Set, no knowledge of fibrations is required to understand it because all constructions are given concretely rather than in their fibrational forms.

We begin by considering what we might naively expect an induction rule for an inductive data type μF to look like. The derivation for *Nat* in Section 2 suggests that, in general, it should look something like this:

$$ind \ : \ \forall P : \mu\, F \to \mathsf{Set}.\ \ ???\ \ \to\ \forall\, x : \mu F.\, P\, x$$

But what should the premises — denoted ??? here — of the generic induction rule *ind* be? Since we want to construct, for any term $x : \mu F$, a proof term of type $P\,x$ from proof terms for x's substructures, and since the functionality of the *fold* operator for μF is precisely to compute a value for $x : \mu F$ from the values for x's substructures, it is natural to try to equip P with an F-algebra structure that can be input to *fold* to yield a mapping of each $x : \mu F$ to an element of $P\,x$. But this approach quickly hits a snag. Since the codomain of every predicate $P : \mu F \to \mathsf{Set}$ is Set itself, rather than an object of Set, F cannot be applied to P as is needed to equip P with an F-algebra structure. Moreover, an induction rule for μF cannot be obtained by applying *fold* to an F-algebra with carrier $P\,x$ for any specific x. This suggests that we should try to construct an F-algebra not for $P\,x$ for each term x, but rather for P itself.

Such considerations led Hermida and Jacobs [10] to define a category of predicates $\mathcal{P}$ and a lifting for each polynomial functor F on Set to a functor $\hat{F}$ on $\mathcal{P}$ that respects the structure of F. They then constructed $\hat{F}$-algebras with carrier $\mathcal{P}$ to serve as the premises of their induction rules. The crucial part of their construction, namely the lifting of polynomial functors, proceeds inductively and includes clauses such as

$$(\widehat{F+G})\ P = \hat{F}P + \hat{G}P$$

and

$$(\widehat{F \times G})\, P = \hat{F} P \times \hat{G} P$$

The construction of Hermida and Jacobs is very general: they consider functors on bicartesian categories rather than just on Set, and represent properties by bicartesian fibrations over such categories instead of using the specific notion of predicate from Definition 3.2 below. On the other hand, they define liftings for polynomial functors.

The construction we give in this section is in some sense orthogonal to Hermida and Jacobs': we focus exclusively on functors on Set and a particular category of predicates, and show how to define liftings for all inductive functors on Set, including non-polynomial ones. In this setting, the induction rule we derive properly extends Hermida and Jacobs', thus catering for a variety of data types that they cannot treat. In the next section we derive analogous results in the general fibrational setting. This allows us to derive sound induction rules for initial algebras of functors defined on categories other than Set which can be used to prove arbitrary properties that are suitably fibred over the category interpreting types.

We begin with the definition of a predicate.

Definition 3.1. Let X be a set. A *predicate on* X is a function $P : X \to \mathsf{Set}$ mapping each $x \in X$ to a set $P\, x$. We call X the *domain* of P.

We may speak simply of "a predicate P" if the domain of P is understood. A predicate P on X can be thought of as mapping each element x of X to the set of proofs that P holds for x. We now define our category of predicates.

Definition 3.2. The *category of predicates* $\mathcal{P}$ has predicates as its objects. A morphism from a predicate $P : X \to \mathsf{Set}$ to a predicate $P' : X' \to \mathsf{Set}$ is a pair $(f, f^{\sim}) : P \to P'$ of functions, where $f : X \to X'$ and $f^{\sim} : \forall x : X.\, P\, x \to P'(f\, x)$. Composition of predicate morphisms is given by $(g, g^{\sim}) \circ (f, f^{\sim}) = (g \circ f,\ \lambda xp.\, g^{\sim}(f x)(f^{\sim} xp))$.

Diagrammatically, we have

$$\begin{tikzcd} X \arrow[rr, "f"] \arrow[dr, "P"'] & \arrow[d, phantom, "\xrightarrow{f^{\sim}}"] & X' \arrow[dl, "P'"] \\ & \mathsf{Set} & \end{tikzcd}$$

As the diagram indicates, the notion of a morphism from P to P' does not require the sets of proofs $P\, x$ and $P'\,(f\, x)$, for any $x \in X$, to be *equal*. Instead, it requires only the existence of a function $f^{\sim}$ which maps, for each x, each proof in $P\, x$ to a proof in $P'\,(f\, x)$. We denote by $U : \mathcal{P} \to \mathsf{Set}$ the *forgetful functor* mapping each predicate $P : X \to \mathsf{Set}$ to its domain X and each predicate morphism $(f, f^{\sim})$ to f.

An alternative to Definition 3.2 would take the category of predicates to be the arrow category over Set, but the natural lifting in this setting does not indicate how to generalise liftings to other fibrations. Indeed, if properties are modelled as functions, then every functor can be applied to a property, and hence every functor can be its own lifting. In the general fibrational setting, however, properties are not necessarily modelled by functions, so a functor cannot, in general, be its own lifting. The decision not to use arrow categories to model properties is thus dictated by our desire to lift functors in a way that indicates how liftings can be constructed in the general fibrational setting.

We can now give a precise definition of a lifting.

Definition 3.3. Let F be a functor on Set. A *lifting* of F from Set to $\mathcal{P}$ is a functor $\hat{F}$ on $\mathcal{P}$ such that the following diagram commutes:

$$\begin{tikzcd} \mathcal{P} \arrow[r, "\hat{F}"] \arrow[d, "U"'] & \mathcal{P} \arrow[d, "U"] \\ \mathsf{Set} \arrow[r, "F"'] & \mathsf{Set} \end{tikzcd}$$

We can decode the definition of $\hat{F}$ as follows. The object part of $\hat{F}$ must map each predicate $P : X \to \mathsf{Set}$ to a predicate $\hat{F}P : F\,X \to \mathsf{Set}$, and thus can be thought of type-theoretically as a function $\forall(X : \mathsf{Set}).\,(X \to \mathsf{Set}) \to F\,X \to \mathsf{Set}$. Of course, $\hat{F}$ must also act on morphisms in a functorial manner.

We can now use the definition of a lifting to derive the standard induction rule from Section 2 for *Nat* as follows.

Example 1. The data type of natural numbers is μN where N is the functor on Set defined by $N\,X = 1 + X$. A lifting $\hat{N}$ of N can be defined by sending each predicate $P : X \to \mathsf{Set}$ to the predicate $\widehat{N}P : NX \to \mathsf{Set}$ given by

$$\begin{array}{lcl} \hat{N}P\,(\mathit{inl}\,\cdot) & = & 1 \\ \hat{N}P\,(\mathit{inr}\,n) & = & P\,n \end{array}$$

An $\hat{N}$-algebra with carrier $P : \mathit{Nat} \to \mathsf{Set}$ can be given by $\mathit{in} : 1 + \mathit{Nat} \to \mathit{Nat}$ and $\mathit{in}^{\sim} : \forall t : 1 + \mathit{Nat}.\,\hat{N}P\,t \to P(\mathit{in}\,t)$. Since $\mathit{in}\,(\mathit{inl}\,\cdot) = 0$ and $\mathit{in}\,(\mathit{inr}\,n) = n+1$, we see that $\mathit{in}^{\sim}$ consists of an element $h_1 : P\,0$ and a function $h_2 : \forall n : \mathit{Nat}.\,P\,n \to P\,(n+1)$. Thus, the second component $\mathit{in}^{\sim}$ of an $\hat{N}$-algebra with carrier $P : \mathit{Nat} \to \mathsf{Set}$ and first component in gives the premises of the familiar induction rule in Example 1.

The notion of predicate comprehension is a key ingredient of our lifting. It begins to explain, abstractly, what the use of Σ-types is in the theory of induction, and is the key construct allowing us to define liftings for non-polynomial, as well as polynomial, functors.

Definition 3.4. Let P be a predicate on X. The *comprehension* of P, denoted $\{P\}$, is the type $\Sigma x : X.\,P\,x$ comprising pairs (x, p) where $x : X$ and $p : Px$. The map taking each predicate P to $\{P\}$, and taking each predicate morphism $(f, f^{\sim}) : P \to P'$ to the morphism $\{(f, f^{\sim})\} : \{P\} \to \{P'\}$ defined by $\{(f, f^{\sim})\}(x, p) = (f\,x, f^{\sim}\,x\,p)$, defines the *comprehension functor* $\{-\}$ from $\mathcal{P}$ to Set.

We are now in a position to define liftings uniformly for all functors:

Definition 3.5. If F is a functor on Set, then the lifting $\hat{F}$ is the functor on $\mathcal{P}$ given as follows. For every predicate P on X, $\hat{F}\,P : F\,X \to \mathsf{Set}$ is defined by $\hat{F}\,P = (F\,\pi_P)^{-1}$, where the natural transformation $\pi : \{-\} \to U$ is given by $\pi_P\,(x, p) = x$. For every predicate morphism $f : P \to P'$, $\hat{F}f = (k, k^{\sim})$ where $k = FUf$, and $k^{\sim} : \forall y : FX.\,\hat{F}P\,y \to \hat{F}P'\,(k\,y)$ is given by $k^{\sim}\,y\,z = F\{f\}z$.

In the above definition, note that the inverse image f^{-1} of $f : X \to Y$ is indeed a predicate $P : Y \to \mathsf{Set}$. Thus if P is a predicate on X, then $\pi_P : \{P\} \to X$ and $F\pi_P : F\{P\} \to FX$. Thus $\hat{F}P$ is a predicate on FX, so $\hat{F}$ is a lifting of F from Set to $\mathcal{P}$. The lifting $\hat{F}$ captures an "all" modality, in that it generalises Haskell's `all` function on lists to arbitrary data types. A similar modality is given in [17] for indexed containers.

The lifting in Example 1 is the instantiation of the construction in Definition 3.5 to the functor $NX = 1 + X$ on Set. Indeed, if P is any predicate, then $\hat{N}\,P = (N\,\pi_P)^{-1}$, i.e., $\hat{N}\,P = (id + \pi_P)^{-1}$. Then, since the inverse image of the coproduct of functions is the coproduct of their inverse images, since $id^{-1}\,1 = 1$, and since $\pi_P^{-1} n = \{(n,p) \mid p : Pn\}$ for all n, we have $\hat{N}\,P\,(inl\,\cdot) = 1$ and $\hat{N}\,P\,(inr\,n) = P\,n$. As we will see, a similar situation to that for *Nat* holds in general: for any functor F on Set, the second component of an $\hat{F}$-algebra whose carrier is the predicate P on the data type μF and whose first component is *in* gives the premises of an induction rule that can be used to show that P holds for all data of type μF.

The rest of this section shows that F-algebras with carrier $\{P\}$ are interderivable with $\hat{F}$-algebras with carrier P, and then uses this result to derive our induction rule.

Definition 3.6. The functor $K_1 : \mathsf{Set} \to \mathcal{P}$ maps each set X to the predicate $K_1X = \lambda x : X.\,1$ on X and each $f : X \to Y$ to the predicate morphism $(f, \lambda x : X.\,id)$.

The predicate K_1X is called the *truth predicate on* X. For every $x : X$, the set K_1Xx of proofs that K_1X holds for x is a singleton, and thus is non-empty. We intuitively think of a predicate $P : X \to \mathsf{Set}$ as being true if Px is non-empty for every $x : X$. We therefore consider P to be true if there exists a predicate morphism from K_1X to P whose first component is id_X. For any functor F, the lifting $\hat{F}$ is *truth-preserving*, i.e., $\hat{F}$ maps the truth predicate on any set X to that on FX.

Lemma 3.7. *For any functor F on Set and any set X, $\hat{F}(K_1X) \cong K_1(FX)$.*

Proof. By Definition 3.5, $\hat{F}(K_1X) = (F\pi_{K_1X})^{-1}$. We have that π_{K_1X} is an isomorphism because there is only one proof of K_1X for each $x : X$, and thus that $F\,\pi_{K_1X}$ is an isomorphism as well. As a result, $(F\,\pi_{K_1X})^{-1}$ maps every $y : FX$ to a singleton set, and therefore $\hat{F}(K_1X) = (F\pi_{K_1X})^{-1} \cong \lambda y : FX.\,1 = K_1(FX)$. □

The fact that K_1 is a left-adjoint to $\{-\}$ is critical to the constructions below. This is proved in [10]; we include its proof here for completeness and to establish notation. The description of comprehension as a right adjoint can be traced back to Lawvere [14].

Lemma 3.8. *K_1 is left adjoint to $\{-\}$.*

Proof. We must show that, for any predicate P and any set Y, the set $\mathcal{P}(K_1Y, P)$ of morphisms from K_1Y to P in $\mathcal{P}$ is in bijective correspondence with the set $\mathsf{Set}(Y, \{P\})$ of morphisms from Y to $\{P\}$ in Set. Define maps $(-)^\dagger : \mathsf{Set}(Y, \{P\}) \to \mathcal{P}(K_1Y, P)$ and $(-)^\# : \mathcal{P}(K_1Y, P) \to \mathsf{Set}(Y, \{P\})$ by $h^\dagger = (h_1, h_2)$ where $hy = (v, p)$, $h_1y = v$ and $h_2y = p$, and $(k, k^\sim)^\# = \lambda(y : Y).\,(ky, k^\sim y)$. These give a natural isomorphism between $\mathsf{Set}(Y, \{P\})$ and $\mathcal{P}(K_1Y, P)$. □

Naturality of $(-)^\dagger$ ensures that $(g \circ f)^\dagger \;=\; g^\dagger \circ K_1 f$ for all $f : Y' \to Y$ and $g : Y \to \{P\}$. Similarly for $(-)^\#$. Moreover, $id^\dagger$ is the counit, at P, of the adjunction between K_1 and $\{-\}$. These observations are used in the proof of Lemma 3.10. Lemmas 3.9 and 3.10 are the key results relating F-algebras and $\hat{F}$ algebras, i.e., relating iteration and induction. They are special cases of Theorem 4.8 below, but we include their proofs to ensure continuity of our presentation and to ensure that this section is self-contained.

We first we show how to construct $\hat{F}$-algebras from F-algebras

Lemma 3.9. *There is a functor $\Phi : Alg_F \to Alg_{\hat{F}}$ such that if $k : FX \to X$, then $\Phi k : \hat{F}(K_1 X) \to K_1 X$.*

Proof. For an F-algebra $k : FX \to X$ define $\Phi k = K_1 k$, and for two F-algebras $k : FX \to X$ and $k' : FX' \to X'$ and an F-algebra morphism $h : X \to X'$ between them define the $\hat{F}$-algebra morphism $\Phi h : \Phi k \to \Phi k'$ by $\Phi h = K_1 h$. Then $K_1(FX) \cong \hat{F}(K_1 X)$ by Lemma 3.7, so that Φk is an $\hat{F}$-algebra and $K_1 h$ is an $\hat{F}$-algebra morphism. It is easy to see that Φ preserves identities and composition. $\square$

We can also construct F-algebras from $\hat{F}$-algebras.

Lemma 3.10. *The functor Φ has a right adjoint Ψ such that if $j : \hat{F}P \to P$, then $\Psi j : F\{P\} \to \{P\}$.*

Proof. We construct the adjoint functor $\Psi : Alg_{\hat{F}} \to Alg_F$ as follows. Given an $\hat{F}$-algebra $j : \hat{F}P \to P$, we use the fact that $\hat{F}(K_1\{P\}) \cong K_1(F\{P\})$ by Lemma 3.7 to define $\Psi j : F\{P\} \to \{P\}$ by $\Psi j = (j \circ \hat{F}\, id^\dagger)^\#$. To specify the action of Ψ on an $\hat{F}$-algebra morphism h, define $\Psi h = \{h\}$. Clearly Ψ preserves identity and composition.

Next we show $\Phi \dashv \Psi$, i.e., for every F-algebra $k : FX \to X$ and $\hat{F}$-algebra $j : \hat{F}P \to P$ with P a predicate on X, there is a natural isomorphism between F-algebra morphisms from k to Ψj and $\hat{F}$-algebra morphisms from Φk to j. We first observe that an F-algebra morphism from k to Ψj is a map from X to $\{P\}$, and an $\hat{F}$-algebra morphism from Φk to j is a map from $K_1 X$ to P. A natural isomorphism between such maps is given by the adjunction $K_1 \dashv \{-\}$ from Lemma 3.8. We must check that $f : X \to \{P\}$ is an F-algebra morphism from k to Ψj iff $f^\dagger : K_1 X \to P$ is an $\hat{F}$-algebra morphism from Φk to j.

To this end, assume $f : X \to \{P\}$ is an F-algebra morphism from k to Ψj, i.e., assume $f \circ k = \Psi j \circ F f$. We must prove that $f^\dagger \circ \Phi k = j \circ \hat{F} f^\dagger$. By the definition of Φ in Lemma 3.9, this amounts to showing $f^\dagger \circ K_1 k = j \circ \hat{F} f^\dagger$. Now, since $(-)^\dagger$ is an isomorphism, f is an F-algebra morphism iff $(f \circ k)^\dagger = (\Psi j \circ F f)^\dagger$. Naturality of $(-)^\dagger$ ensures that $(f \circ k)^\dagger = f^\dagger \circ K_1 k$ and that $(\Psi j \circ F f)^\dagger = (\Psi j)^\dagger \circ K_1(F f)$, so the previous equality holds iff

$$f^\dagger \circ K_1 k \;=\; (\Psi j)^\dagger \circ K_1(F f) \tag{3.1}$$

But

$$\begin{array}{lll}
 & j \circ \hat{F} f^\dagger & \\
= & j \circ \hat{F} id^\dagger \circ K_1 f) & \text{by naturality of } (-)^\dagger \text{ and } f = id \circ f \\
= & (j \circ \hat{F}\, id^\dagger) \circ \hat{F}(K_1 f) & \text{by the functoriality of } \hat{F} \\
= & (\Psi j)^\dagger \circ K_1(F f) & \text{by the definition of } \Psi,\text{ the fact that } (-)^\dagger \text{ and } (-)^\# \\
 & & \quad \text{are inverses, and Lemma 3.7} \\
= & f^\dagger \circ K_1 k & \text{by Equation 3.1}
\end{array}$$

Thus, $f^\dagger$ is indeed an $\hat{F}$-algebra morphism from Φk to j. $\square$

Lemma 3.10 ensures that F-algebras with carrier $\{P\}$ are interderivable with $\hat{F}$-algebras with carrier P. For example, the N-algebra $[\alpha, \beta]$ with carrier $\{P\}$ from Section 2 can be derived from the $\hat{N}$-algebra with carrier P given in Example 1. Since we define a lifting $\hat{F}$ for any functor F, Lemma 3.10 thus shows how to construct F-algebras with carrier $\Sigma x : \mu F.\, P x$ for any functor F and predicate P on μF.

Corollary 3.11. *For any functor F on* Set*, the predicate $K_1(\mu F)$ is the carrier of the initial $\hat{F}$-algebra.*

Proof. Since Φ is a left adjoint it preserves initial objects, so applying Φ to the initial F-algebra $in : F(\mu F) \to \mu F$ gives the initial $\hat{F}$-algebra. By Lemma 3.9, $\Phi\, in$ has type $\hat{F}(K_1(\mu F)) \to K_1(\mu F)$, so the carrier of the initial $\hat{F}$-algebra is $K_1(\mu F)$. $\square$

We can now derive our generic induction rule. For every predicate P on X and every $\hat{F}$-algebra $(k, k^\sim) : \hat{F}P \to P$, Lemma 3.10 ensures that Ψ constructs from $(k, k^\sim)$ an F-algebra with carrier $\{P\}$. Applying the iteration operator to this algebra gives a map

$$\mathit{fold}\ (\Psi\,(k, k^\sim)) : \mu F \to \{P\}$$

This map decomposes into two parts: $\phi = \pi_P \circ \mathit{fold}\ (\Psi\,(k, k^\sim)) : \mu F \to X$ and $\psi : \forall (t : \mu F).\, P\,(\phi\, t)$. Initiality of $in : F(\mu F) \to \mu F$, the definition of Ψ, and the naturality of π_P ensure $\phi = \mathit{fold}\, k$. Recalling that π'_P is the second projection on dependent pairs involving the predicate P, this gives the following sound generic induction rule for the type X, which reduces induction to iteration:

$$\begin{array}{lcl} \mathit{genind} & : & \forall\,(F : \mathsf{Set} \to \mathsf{Set})\ (P : X \to \mathsf{Set})\ ((k, k^\sim) : (\hat{F}P \to P))\ (t : \mu F). \\ & & \quad P\,(\mathit{fold}\, k\, t) \\ \mathit{genind}\, F\, P & = & \pi'_P \circ \mathit{fold} \circ \Psi \end{array}$$

Notice this induction rule is actually capable of dealing with predicates over arbitrary sets and not just predicates over μF. However, when $X = \mu F$ and $k = in$, initiality of in further ensures that $\phi = \mathit{fold}\, in = id$, and thus that *genind* specialises to the expected induction rule for an inductive data type μF:

$$\begin{array}{lcl} \mathit{ind} & : & \forall\,(F : \mathsf{Set} \to \mathsf{Set})\ (P : \mu F \to \mathsf{Set})\ ((k, k^\sim) : (\hat{F}P \to P)). \\ & & \quad (k = in) \to \forall (t : \mu F).\, P\, t \\ \mathit{ind}\, F\, P & = & \pi'_P \circ \mathit{fold} \circ \Psi \end{array}$$

This rule can be instantiated to familiar rules for polynomial data types, as well as to ones we would expect for data types such as rose trees and finite hereditary sets, both of which lie outside the scope of Hermida and Jacobs' method.

Example 2. The data type of rose trees is given in Haskell-like syntax by

$$\mathit{data\ Rose} = \mathit{Node}(\mathit{List\ Rose})$$

The functor underlying *Rose* is $FX = \mathit{List}\, X$ and its induction rule is

$$\begin{array}{lcl} \mathit{indRose} & : & \forall\ (P : \mathit{Rose} \to \mathsf{Set})\ ((k, k^\sim) : (\hat{F}P \to P)). \\ & & \quad (k = in) \to \forall (x : \mathit{Rose}).\, P\, x \\ \mathit{indRose}\, F\, P & = & \pi'_P \circ \mathit{fold} \circ \Psi \end{array}$$

Calculating $\hat{F}P = (F\pi_P)^{-1} : F\,\mathit{Rose} \to \mathsf{Set}$, and writing $xs\, !!\, k$ for the k^{th} component of a list xs, we have that

$$\begin{array}{ll} & \hat{F}\, P\, rs \\ = & \{z : F\{P\} \mid F\, \pi_p\, z = rs\} \\ = & \{cps : \mathit{List}\,\{P\} \mid \mathit{List}\ \pi_P\ cps = rs\} \\ = & \{cps : \mathit{List}\,\{P\} \mid \forall k < \mathit{length}\, cps.\, \pi_P\,(cps\, !!\, k) = rs\, !!\, k\} \end{array}$$

An $\hat{F}$-algebra whose underlying F-algebra is $in : F\,Rose \to Rose$ is thus a pair of functions $(in, k^{\sim})$, where $k^{\sim}$ has type

$$\begin{array}{rl} = & \forall rs : List\,Rose. \\ & \quad \{cps : List\,\{P\} \mid \forall k < length\,cps.\, \pi_P\,(cps\,!!\,k) = rs\,!!\,k\} \;\to\; P\,(Node\,rs) \\ = & \forall rs : List\,Rose.\,(\forall k < length\;rs.\,P\,(rs\,!!\,k)) \to P(Node\,rs) \end{array}$$

The last equality is due to surjective pairing for dependent products and the fact that $length\,cps = length\,rs$. The type of $k^{\sim}$ gives the hypotheses of the induction rule for rose trees.

Although finite hereditary sets are defined in terms of quotients, and thus lie outside the scope of previously known methods, they can be treated with ours.

Example 3. Hereditary sets are sets whose elements are themselves sets, and so are the core data structures within set theory. The data type HS of finitary hereditary sets is μP_f for the finite powerset functor P_f. We can derive an induction rule for finite hereditary sets as follows. If $P : X \to \mathsf{Set}$, then $P_f \pi_P : P_f(\Sigma x : X.Px) \to P_f X$ maps each set $\{(x_1, p_1), \ldots, (x_n, p_n)\}$ to the set $\{x_1, \ldots, x_n\}$, so that $(P_f \pi_P)^{-1}$ maps a set $\{x_1, \ldots, x_n\}$ to the set $Px_1 \times \ldots \times Px_n$. A $\hat{P}_f$-algebra with carrier $P : HS \to \mathsf{Set}$ and first component in therefore has as its second component a function of type

$$\forall(\{s_1, \ldots, s_n\} : P_f(HS)).\, Ps_1 \times \ldots \times Ps_n \to P(in\{s_1, \ldots, s_n\})$$

The induction rule for finite hereditary sets is thus

$$\begin{array}{l} indHS :: (\forall(\{s_1, \ldots, s_n\} : P_f(HS)).\, Ps_1 \times \ldots \times Ps_n \to P(in\{s_1, \ldots, s_n\})) \\ \qquad\qquad \to \forall(s : HS).P\,s \end{array}$$

4. Generic Fibrational Induction Rules

We can treat more general notions of predicates using fibrations. We motivate the use of fibrations by observing that i) the semantics of data types in languages involving recursion and other effects usually involves categories other than Set; ii) in such circumstances, the notion of a predicate can no longer be taken as a function with codomain Set; and iii) even when working in Set there are reasonable notions of "predicate" other than that in Section 3. (For example, a predicate on a set X could be a subobject of X). Moreover, when, in future work, we consider induction rules for more sophisticated classes of data types such as indexed containers, inductive families, and inductive recursive families (see Section 5), we will not want to have to develop an individual *ad hoc* theory of induction for each such class. Instead, we will want to appropriately instantiate a single generic theory of induction. That is, we will want a uniform axiomatic approach to induction that is widely applicable, and that abstracts over the specific choices of category, functor, and predicate giving rise to different forms of induction for specific classes of data types.

Fibrations support precisely such an axiomatic approach. This section therefore generalises the constructions of the previous one to the general fibrational setting. The standard model of type theory based on locally cartesian closed categories does arise as a specific fibration — namely, the codomain fibration over Set — and this fibration is equivalent to the families fibration over Set. But the general fibrational setting is far more flexible. Moreover, in locally cartesian closed models of type theory, predicates and types coexist in the same category, so that each functor can be taken to be its own lifting. In the general fibrational setting, predicates are not simply functions or morphisms, properties and types do not coexist in the same category, and a functor cannot be taken to be its own lifting. There is no choice but to construct a lifting from scratch. A treatment of induction based solely on locally cartesian closed categories would not, therefore, indicate how to treat induction in more general fibrations.

Another reason for working in the general fibrational setting is that this facilitates a direct comparison of our work with that of Hermida and Jacobs [10]. This is important, since their approach is the most closely related to ours. The main difference between their approach and ours is that they use fibred products and coproducts to define provably sound induction rules for polynomial functors, whereas we use left adjoints to reindexing functors to define provably sound induction rules for *all* inductive functors. In this section we consider situations when both approaches are possible and give mild conditions under which our results coincide with theirs when restricted to polynomial functors.

The remainder of this section is organised as follows. In Section 4.1 we recall the definition of a fibration, expand and motivate this definition, and fix some basic terminology surrounding fibrations. We then give some examples of fibrations, including the families fibration over Set, the codomain fibration, and the subobject fibration. In Section 4.2 we recall a useful theorem from [10] that indicates when a truth-preserving lifting of a functor to a category of predicates has an initial algebra. This is the key theorem used to prove the soundness of our generic fibrational induction rule. In Section 4.3 we construct truth-preserving liftings for all inductive functors. We do this first in the codomain fibration, and then, using intuitions from its presentation as the families fibration over Set, as studied in Section 3, in a general fibrational setting. Finally, in Section 4.4 we establish a number of properties of the liftings, and hence of the induction rules, that we have derived. In particular, we characterise the lifting that generates our induction rules.

4.1. **Fibrations in a Nutshell.** In this section we recall the notion of a fibration. More details about fibrations can be found in, e.g., [12, 20]. We begin with an auxiliary definition.

Definition 4.1. Let $U : \mathcal{E} \to \mathcal{B}$ be a functor.

(1) 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 $g \circ h = g'$.
(2) A morphism $g : P \to Q$ in $\mathcal{E}$ is *opcartesian* over a morphism $f : X \to Y$ in $\mathcal{B}$ if $Ug = f$, and for every $g' : P \to Q'$ in $\mathcal{E}$ for which $Ug' = v \circ f$ for some $v : Y \to UQ'$ there exists a unique $h : Q \to Q'$ in $\mathcal{E}$ such that $Uh = v$ and $h \circ g = g'$.

It is not hard to see that the cartesian morphism $f^{\S}_P$ over a morphism f with codomain UP is unique up to isomorphism, and similarly for the opcartesian morphism $f_{\S}^P$. If P is an object of $\mathcal{E}$, then we write f^*P for the domain of $f^\S_P$ and $\Sigma_f P$ for the codomain of $f^P_\S$. We can capture cartesian and opcartesian morphisms diagrammatically as follows.

$$\begin{tikzcd}
\mathcal{E} \arrow[ddd, "U"'] & Q' \arrow[d, dotted, "h"'] \arrow[dr, "g'"] & & & Q' \\
& f^*P \arrow[r, "f^\S_P"'] & P & P \arrow[ur, "g'"] \arrow[r, "f^P_\S"'] & \Sigma_f P \arrow[u, dotted, "h"'] \\
& UQ' \arrow[d, "v"'] \arrow[dr, "Ug'"] & & & UQ' \\
\mathcal{B} & X \arrow[r, "f"'] & Y & X \arrow[ur, "Ug'"] \arrow[r, "f"'] & Y \arrow[u, "v"']
\end{tikzcd}$$

Cartesian morphisms (opcartesian morphisms) are the essence of fibrations (resp., opfibrations). We introduce both fibrations and their duals now since the latter will prove useful later in our development. Below we speak primarily of fibrations, with the understanding that the dual observations hold for opfibrations.

Definition 4.2. Let $U : \mathcal{E} \to \mathcal{B}$ be a functor. Then U 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 $f^\S_P : Q \to P$ in $\mathcal{E}$ above f. Similarly, 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 $f^P_\S : P \to Q$ in $\mathcal{E}$ above f. A functor U a *bifibration* if it is simultaneously a fibration and an opfibration.

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 the total category $\mathcal{E}$ can be thought of as properties, objects of the base category $\mathcal{B}$ can be thought of as types, and U can be thought of as mapping each property P in $\mathcal{E}$ to the type UP of which P is a property. One fibration U can capture many different properties of the same type, so U is not injective on objects. We say that an object P in $\mathcal{E}$ is *above* its image UP under U, and similarly for morphisms. For any object X of $\mathcal{B}$, we write $\mathcal{E}_X$ for the *fibre above* X, i.e., for the subcategory of $\mathcal{E}$ consisting of objects above X and morphisms above id. 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 $f^* : \mathcal{E}_Y \to \mathcal{E}_X$. Indeed, for each morphism $k : P \to P'$ in $\mathcal{E}_Y$, f^*k is the morphism satisfying $k \circ f^\S_P = f^\S_{P'} \circ f^*k$. The universal property of $f^\S_{P'}$ ensures the existence and uniqueness of f^*k. We call the functor f^* the *reindexing functor induced by* f. A similar situation ensures for opfibrations, and we call the functor $\Sigma_f : \mathcal{E}_X \to \mathcal{E}_Y$ which extends the function mapping each object P of $\mathcal{E}$ to $\Sigma_f P$ the *opreindexing functor*.

Example 4. The functor $U : \mathcal{P} \to \mathsf{Set}$ defined in Section 3 is called the *families fibration over* Set. Given a function $f : X \to Y$ and a predicate $P : Y \to \mathsf{Set}$ we can define a cartesian map $f^\S_P$ whose domain f^*P is $P \circ f$, and which comprises the pair $(f, \lambda x : X.\, id)$. The fibre $\mathcal{P}_X$ above a set X has predicates $P : X \to \mathsf{Set}$ as its objects. A morphism in $\mathcal{P}_X$ from $P : X \to \mathsf{Set}$ to $P' : X \to \mathsf{Set}$ is a function of type $\forall x : X.\, Px \to P'x$.

Example 5. Let $\mathcal{B}$ be a category. The *arrow category* of $\mathcal{B}$, denoted $\mathcal{B}^{\to}$, has the morphisms, or arrows, 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 the following diagram commutes:

$$\begin{tikzcd} X \arrow[r, "\alpha_1"] \arrow[d, "f"'] & X' \arrow[d, "f'"] \\ Y \arrow[r, "\alpha_2"'] & Y' \end{tikzcd}$$

i.e., such that $\alpha_2 \circ f = f' \circ \alpha_1$. It is easy to see that this definition indeed gives a category.

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 $\mathcal{B}$ and a morphism (α_1, α_2) of $\mathcal{B}^{\to}$ to α_2. If $\mathcal{B}$ has pullbacks, then cod is a fibration, called the *codomain fibration over* $\mathcal{B}$. Indeed, given an object $f : X \to Y$ in the fibre above Y and a morphism $f' : X' \to Y$ in $\mathcal{B}$, the pullback of f along f' gives a cartesian morphism above f' as required. The fibre above an object Y of $\mathcal{B}$ has those morphisms of $\mathcal{B}$ that map into Y as its objects. A morphism in $(\mathcal{B}^{\to})_Y$ from $f : X \to Y$ to $f' : X' \to Y$ is a morphism $\alpha_1 : X \to X'$ in $\mathcal{B}$ such that $f = f' \circ \alpha_1$.

Example 6. If $\mathcal{B}$ is a category, then the category of subobjects of $\mathcal{B}$, denoted $Sub(\mathcal{B})$, has 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 pair of morphisms (α_1, α_2) in $\mathcal{B}$ such that $\alpha_2 \circ f = f' \circ \alpha_1$.

The map $U : Sub(\mathcal{B}) \to \mathcal{B}$ sending a subobject $f : X \hookrightarrow Y$ to Y extends to a functor. If $\mathcal{B}$ has pullbacks, then U is a fibration, called the *subobject fibration over* $\mathcal{B}$; indeed, pullbacks again give cartesian morphisms since the pullback of a monomorphism is a monomorphism. The fibre above an object Y of $\mathcal{B}$ has as objects the subobjects of Y. A morphism in $Sub(\mathcal{B})_Y$ from $f : X \hookrightarrow Y$ to $f' : X' \hookrightarrow Y$ is a map $\alpha_1 : X \to X'$ in $\mathcal{B}$ such that $f = f' \circ \alpha_1$. If such a morphism exists then it is, of course, unique.

4.2. **Lifting, Truth, and Comprehension.** We now generalise the notions of lifting, truth, and comprehension to the general fibrational setting. We prove that, in such a setting, if an inductive functor has a truth-preserving lifting, then its lifting is also inductive. We then see that inductiveness of the lifted functor is sufficient to guarantee the soundness of our generic fibrational induction rule. This subsection is essentially our presentation of pre-existing results from [10]. We include it because it forms a natural part of our narrative, and because simply citing the material would hinder the continuity of our presentation.

Recall from Section 3 that the first step in deriving an induction rule for a datatype interpreted in Set is to lift the functor whose fixed point the data type is to the category $\mathcal{P}$ of predicates. More specifically, in Definition 3.3 we defined a lifting of a functor $F : \mathsf{Set} \to \mathsf{Set}$ to be a functor $\hat{F} : \mathcal{P} \to \mathcal{P}$ such that $U\hat{F} = FU$. We can use these observations to generalise the notion of a lifting to the fibrational setting as follows.

Definition 4.3. Let $U : \mathcal{E} \to \mathcal{B}$ be a fibration and F be a functor on $\mathcal{B}$. A *lifting* of F with respect to U is a functor $\hat{F} : \mathcal{E} \to \mathcal{E}$ such that the following diagram commutes:

$$\begin{tikzcd} \mathcal{E} \arrow[r, "\hat{F}"] \arrow[d, "U"'] & \mathcal{E} \arrow[d, "U"] \\ \mathcal{B} \arrow[r, "F"'] & \mathcal{B} \end{tikzcd}$$

In Section 3 we saw that if $P : X \to \mathsf{Set}$ is a predicate over X, then $\hat{F}P$ is a predicate over FX. The analogous result for the general fibrational setting observes that if $\hat{F}$ is a lifting of F and X is an object of $\mathcal{B}$, then $\hat{F}$ restricts to a functor from $\mathcal{E}_X$ to $\mathcal{E}_{FX}$.

By analogy with our results from Section 3, we further expect that the premises of a fibrational induction rule for a datatype μF interpreted in $\mathcal{B}$ should constitute an $\hat{F}$-algebra on $\mathcal{E}$. But in order to construct the conclusion of such a rule, we need to understand how to axiomatically state that a predicate is true. In Section 3, a predicate $P : X \to \mathsf{Set}$ is considered true if there is a morphism in $\mathcal{P}$ from K_1X, the truth predicate on X, to P that is over id_X. Since the mapping of each set X to K_1X is the action on objects of the truth functor $K_1 : \mathsf{Set} \to \mathcal{P}$ (cf. Definition 3.6), we actually endeavour to model the truth functor for the families fibration over Set axiomatically in the general fibrational setting.

Modeling the truth functor axiomatically amounts to understanding its universal property. Since the truth functor in Definition 3.6 maps each set X to the predicate $\lambda x : x.\, 1$, for any set X there is therefore exactly one morphism in the fibre above X from any predicate P over X to $K_1\, X$. This gives a clear categorical description of $K_1\, X$ as a terminal object of the fibre above X and leads, by analogy, to the following definition.

Definition 4.4. Let $U : \mathcal{E} \to \mathcal{B}$ be a fibration. Assume further that, for every object X of $\mathcal{B}$, the fibre $\mathcal{E}_X$ has a terminal object $K_1\, X$ such that, for any $f : X' \to X$ in $\mathcal{B}$, $f^*(K_1\, X) \cong K_1\, X'$. Then the assignment sending each object X in $\mathcal{B}$ to K_1X in $\mathcal{E}$, and each morphism $f : X' \to X$ in $\mathcal{B}$ to the morphism $f^{\S}_{K_1 X}$ in $\mathcal{E}$ defines the (fibred) *truth functor* $K_1 : \mathcal{B} \to \mathcal{E}$.

The (fibred) truth functor is sometimes called the (fibred) *terminal object functor*. With this definition, we have the following standard result:

Lemma 4.5. *K_1 is a (fibred) right adjoint for U.*

The interested reader may wish to consult the literature on fibrations for the definition of a fibred adjunction, but a formal definition will not be needed here. Instead, we can simply stress that a fibred adjunction is first and foremost an adjunction, and then observe that the counit of this adjunction is the identity, so that $UK_1 = Id$. Moreover, K_1 is full and faithful. One simple way to guarantee that a fibration has a truth functor is to assume that both $\mathcal{E}$ and $\mathcal{B}$ have terminal objects and that U maps a terminal object of $\mathcal{E}$ to a terminal object of $\mathcal{B}$. In this case, the fact that reindexing preserves fibred terminal objects ensures that every fibre of $\mathcal{E}$ indeed has a terminal object.

The second fundamental property of liftings used in Section 3 is that they are truth-preserving. This property can now easily be generalised to the general fibrational setting (cf. Definition 3.7).

Definition 4.6. Let $U : \mathcal{E} \to \mathcal{B}$ be a fibration with a truth functor $K_1 : \mathcal{B} \to \mathcal{E}$, let F be a functor on $\mathcal{B}$, and let $\hat{F} : \mathcal{E} \to \mathcal{E}$ be a lifting of F. We say that $\hat{F}$ is a *truth-preserving* lifting of F if, for any object X of $\mathcal{B}$, we have $\hat{F}(K_1X) \cong K_1(FX)$.

The final algebraic structure we required in Section 3 was a comprehension functor $\{-\} : \mathcal{P} \to \mathsf{Set}$. To generalise the comprehension functor to the general fibrational setting we simply note that its universal property is that it is right adjoint to the truth functor K_1 (cf. Definition 3.8). We single out for special attention those fibrations whose truth functors have right adjoints.

Definition 4.7. Let $U : \mathcal{E} \to \mathcal{B}$ be a fibration with a truth functor $K_1 : \mathcal{B} \to \mathcal{E}$. Then U is a *comprehension category with unit* if K_1 has a right adjoint.

If $U : \mathcal{E} \to \mathcal{B}$ is a comprehension category with unit, then we call the right adjoint to K_1 the *comprehension functor* and denote it by $\{-\} : \mathcal{E} \to \mathcal{B}$. With this machinery in place, Hermida and Jacobs [10] show that if U is a comprehension category with unit and $\hat{F}$ is a truth-preserving lifting of F, then $\hat{F}$ is inductive if F is and, in this case, the carrier $\mu\hat{F}$ of the initial $\hat{F}$-algebra is $K_1(\mu F)$. This is proved as a corollary to the following more abstract theorem.

Theorem 4.8. *Let $F : \mathcal{B} \to \mathcal{B}$, $G : \mathcal{A} \to \mathcal{A}$, and $S : \mathcal{B} \to \mathcal{A}$ be functors. A natural transformation $\alpha : GS \to SF$, i.e., a natural transformation α such that*

$$\begin{tikzcd} \mathcal{A} \arrow[r, "G"] & \mathcal{A} \\ \mathcal{B} \arrow[u, "S"] \arrow[r, "F"'] \arrow[ur, phantom, "\searrow\alpha"] & \mathcal{B} \arrow[u, "S"'] \end{tikzcd}$$

induces a functor

$$Alg_F \xrightarrow{\Phi} Alg_G$$

given by $\Phi\ (f : FX \to X) = S\,f \circ \alpha_X$. Moreover, if α is an isomorphism, then a right adjoint T to S induces a right adjoint

$$\begin{tikzcd} Alg_F \arrow[r, bend left, "\Phi"] \arrow[r, phantom, "\top"] & Alg_G \arrow[l, bend left, "\Psi"] \end{tikzcd}$$

given by $\Psi(g : GX \to X) = Tg \circ \beta_X$, where $\beta : FT \to TG$ is the image of $G\,\epsilon \circ \alpha_T^{-1} : SFT \to G$ under the adjunction isomorphism $Hom(S\,X, Y) \cong Hom(X, T\,Y)$, and $\epsilon : ST \to id$ is the counit of this adjunction.

We can instantiate Theorem 4.8 to generalise Lemmas 3.9 and 3.10.

Theorem 4.9. *Let $U : \mathcal{E} \to \mathcal{B}$ be a comprehension category with unit and $F : \mathcal{B} \to \mathcal{B}$ be a functor. If F has a truth-preserving lifting $\hat{F}$ then there is an adjunction $\Phi \dashv \Psi : Alg_F \to Alg_{\hat{F}}$. Moreover, if $f : FX \to X$ then $\Phi f : \hat{F}(K_1X) \to K_1X$, and if $g : \hat{F}P \to P$ then $\Psi g : F\{P\} \to \{P\}$.*

Proof. We instantiate Theorem 4.8, letting $\mathcal{E}$ be $\mathcal{A}$, $\hat{F}$ be G, and K_1 be S. Then α is an isomorphism since $\hat{F}$ is truth-preserving, and we also have that $K_1 \dashv \{-\}$. The theorem thus ensures that Φ maps every F-algebra $f : FX \to X$ to an $\hat{F}$-algebra $\Phi f : \hat{F}(K_1X) \to K_1X$, and that Ψ maps every $\hat{F}$-algebra $g : \hat{F}P \to P$ to an F-algebra $\Psi g : F\{P\} \to \{P\}$. □

Corollary 4.10. *Let $U : \mathcal{E} \to \mathcal{B}$ be a comprehension category with unit and $F : \mathcal{B} \to \mathcal{B}$ be a functor which has a truth-preserving lifting $\hat{F}$. If F is inductive, then so is $\hat{F}$. Moreover, $\mu\hat{F} = K_1(\mu F)$.*

Proof. The hypotheses of the corollary place us in the setting of Theorem 4.9. This theorem guarantees that Φ maps the initial F-algebra $in_F : F(\mu F) \to \mu F$ to an $\hat{F}$-algebra with carrier $K_1(\mu F)$. But since left adjoints preserve initial objects, we must therefore have that the initial $\hat{F}$-algebra has carrier $K_1(\mu F)$. Thus, $\mu\hat{F}$ exists and is isomorphic to $K_1(\mu F)$. □

Theorem 4.11. *Let $U : \mathcal{E} \to \mathcal{B}$ be a comprehension category with unit and $F : \mathcal{B} \to \mathcal{B}$ be an inductive functor. If F has a truth-preserving lifting $\hat{F}$, then the following generic fibrational induction rule is sound:*

$$\begin{array}{lcl} \mathit{genfibind} & : & \forall\,(F : \mathcal{B} \to \mathcal{B})\ (P : \mathcal{E}).\ (\hat{F}\,P \to P) \to (\mu\hat{F} \to P) \\ \mathit{genfibind}\,F\,P & = & \mathit{fold} \end{array}$$

An alternative presentation of *genfibind* is

$$\begin{array}{lcl} \mathit{genfibind} & : & \forall\,(F : \mathcal{B} \to \mathcal{B})\ (P : \mathcal{E}).\ (\hat{F}\,P \to P) \to (\mu F \to \{P\}) \\ \mathit{genfibind}\,F\,P & = & \mathit{fold} \circ \Psi \end{array}$$

We call *genfibind* F the *generic fibrational induction rule* for μF.

In summary, we have generalised the generic induction rule for predicates over Set presented in Section 3 to give a sound generic induction rule for comprehension categories with unit. Our only assumption is that if we start with an inductive functor F on the base of the comprehension category, then there must be a truth-preserving lifting of that functor to the total category of the comprehension category. In that case, we can specialise *genfibind* to get a fibrational induction rule for any datatype μF that can be interpreted in the fibration's base category.

The generic fibrational induction rule *genfibind* does, however, look slightly different from the generic induction rule for set-valued predicates. This is because, in Section 3, we used our knowledge of the specific structure of comprehensions for set-valued predicates to extract proofs for particular data elements from them. But in the fibrational setting, predicates, and hence comprehensions, are left abstract. We therefore take the return type of the general induction scheme *genfibind* to be a comprehension with the expectation that, when the general theory of this section is instantiated to a particular fibration of interest, it may be possible to use knowledge about that fibration to extract from the comprehension constructed by *genfibind* further proof information relevant to the application at hand.

As we have previously mentioned, Hermida and Jacobs provide truth-preserving liftings only for polynomial functors. In Section 4.3, we define a generic truth-preserving lifting for *any* inductive functor on the base category of any fibration which, in addition to being a comprehension category with unit, has left adjoints to all reindexing functors. This gives a sound generic fibrational induction rule for the datatype μF for any functor F on the base category of any such fibration.

4.3. **Constructing Truth-Preserving Liftings.** In light of the previous subsection, it is natural to ask whether or not truth-preserving liftings exist. If so, are they unique? Or, if there are many truth-preserving liftings, is there a specific truth-preserving lifting to choose above others? Is there, perhaps, even a universal truth-preserving lifting? We can also ask about the algebraic structure of liftings. For example, do truth-preserving liftings preserve sums and products of functors?

Answers to some of these questions were given by Hermida and Jacobs, who provided truth-preserving liftings for polynomial functors. To define such liftings they assume that the total category and the base category of the fibration in question have products and coproducts, and that the fibration preserves them. Under these conditions, liftings for polynomial functors can be defined inductively. In this section we go beyond the results of Hermida and Jacobs and construct truth-preserving liftings for *all* inductive functors. We employ a two-stage process, first building truth-preserving liftings under the assumption that the fibration of interest is a codomain fibration, and then using the intuitions of Section 3 to extend this lifting to a more general class of fibrations. In Section 4.4 we consider the questions from the previous paragraph about the algebraic structure of liftings.

4.3.1. *Truth-Preserving Liftings for Codomain Fibrations.* Recall from Example 5 that if $\mathcal{B}$ has pullbacks, then the codomain fibration over $\mathcal{B}$ is the functor $cod : \mathcal{B}^{\rightarrow} \rightarrow \mathcal{B}$. Given a functor $F : \mathcal{B} \rightarrow \mathcal{B}$, it is trivial to define a lifting $F^{\rightarrow} : \mathcal{B}^{\rightarrow} \rightarrow \mathcal{B}^{\rightarrow}$ for this fibration. We can define the functor $F^{\rightarrow}$ to map an object $f : X \rightarrow Y$ of $\mathcal{B}^{\rightarrow}$ to $Ff : FX \rightarrow FY$, and to map a morphism (α_1, α_2) to the morphism $(F\alpha_1, F\alpha_2)$. That $F^{\rightarrow}$ is a lifting is easily verified.

If we further verify that codomain fibrations are comprehension categories with unit, and that the lifting $F^{\rightarrow}$ is truth-preserving, then Theorem 4.11 can be applied to them. For the former, we first observe that the functor $K_1 : \mathcal{B} \rightarrow \mathcal{B}^{\rightarrow}$ mapping an object X to id and a morphism $f : X \rightarrow Y$ to (f, f) is a truth functor for this fibration. (In fact, we can take any isomorphism into X as K_1X; we will use this observation below.) If we let $\mathcal{B}^{\rightarrow}(U, V)$ denote the set of morphisms from an object U to an object V in $\mathcal{B}^{\rightarrow}$, then the fact that K_1 is right adjoint to cod can be established via the natural isomorphism

$$\mathcal{B}^{\rightarrow}(f : X \rightarrow Y,\, K_1Z) = \{(\alpha_1 : X \rightarrow Z,\, \alpha_2 : Y \rightarrow Z) \mid \alpha_1 = \alpha_2 \circ f\} \cong \mathcal{B}(Y, Z) = \mathcal{B}(cod\, f,\, Z)$$

We next show that the functor $dom : \mathcal{B}^{\rightarrow} \rightarrow \mathcal{B}$ mapping an object $f : X \rightarrow Y$ of $\mathcal{B}^{\rightarrow}$ to X and a morphism (α_1, α_2) to α_1 is a comprehension functor for the codomain fibration. That dom is right adjoint to K_1 is established via the natural isomorphism

$$\mathcal{B}^{\rightarrow}(K_1Z,\, f : X \rightarrow Y) = \{(\alpha_1 : Z \rightarrow X, \alpha_2 : Z \rightarrow Y) \mid \alpha_2 = f \circ \alpha_1\} \cong \mathcal{B}(Z, X) = \mathcal{B}(Z, dom\, f)$$

Finally, we have that $F^{\rightarrow}$ is truth-preserving because

$$F^{\rightarrow}(K_1Z) = F^{\rightarrow}\, id = F\, id = id = K_1(FZ)$$

A lifting is implicitly given in [16] for functors on a category with display maps. Such a category is a subfibration of the codomain fibration over that category, and the lifting given there is essentially the lifting for the codomain fibration restricted to the subfibration in question.

4.3.2. *Truth-Preserving Liftings for the Families Fibration over* Set. In Section 3 we defined, for every functor $F : \mathsf{Set} \rightarrow \mathsf{Set}$, a lifting $\hat{F}$ which maps the predicate P to $(F\pi_P)^{-1}$. Looking closely, we realise this lifting decomposes into three parts. Given a predicate P, we first consider the projection function $\pi_P : \{P\} \rightarrow UP$. Next, we apply the functor F to π_P to obtain $F\pi_P : F\{P\} \rightarrow FUP$. Finally, we take the inverse image of $F\pi_P$ to get a predicate over FUP as required.

Note that π is the functor from $\mathcal{P}$ to $\mathsf{Set}^{\rightarrow}$ which maps a predicate P to the projection function $\pi_P : \{P\} \rightarrow UP$ (and maps a predicate morphism $(f, f^{\sim})$ from a predicate $P : X \rightarrow \mathsf{Set}$ to $P' : X' \rightarrow \mathsf{Set}$ to the morphism $(\{(f, f^{\sim})\},\ f)$ from π_P to $\pi_{P'}$; cf. Definition 3.4). If $I : \mathsf{Set}^{\rightarrow} \rightarrow \mathcal{P}$ is the functor sending a function $f : X \rightarrow Y$ to its "inverse" predicate f^{-1} (and a morphism (α_1, α_2) to the predicate morphism $(\alpha_2, \forall y : Y.\ \lambda x : f^{-1}y.\ \alpha_1 x)$), then each of the three steps of defining $\hat{F}$ is functorial and the relationships indicated by the following diagram hold:

$$
\begin{tikzcd}
\mathcal{P} \arrow[rr, bend left, "\pi"] \arrow[dr, "U"'] & & \mathsf{Set}^{\rightarrow} \arrow[ll, bend left, "I"] \arrow[ll, phantom, "\top"] \arrow[dl, "cod"] \\
& \mathsf{Set} &
\end{tikzcd}
$$

Note that the adjunction $I \dashv \pi$ is an equivalence. This observation is not, however, necessary for our subsequent development; in particular, it is not needed for Theorem 4.14.

The above presentation of the lifting $\hat{F}$ of a functor F for the families fibration over Set uses the lifting of F for the codomain fibration over Set. Indeed, writing $F^{\rightarrow}$ for the lifting of F for the codomain fibration over Set, we have that $\hat{F} = IF^{\rightarrow}\pi$. Moreover, since π and I are truth-preserving (see the proof of Lemma 3.7), and since we have already seen that liftings for codomain fibrations are truth-preserving, we have that $\hat{F}$ is truth-preserving because each of its three constituent functors is. Finally, since we showed in Section 3 that the families fibration over Set is a comprehension category with unit, Theorem 4.11 can be applied to it.

Excitingly, as we shall see in the next subsection, the above presentation of the lifting of a functor for the families fibration over Set generalises to many other fibrations!

4.3.3. *Truth-Preserving Liftings for Other Fibrations.* We now turn our attention to the task of constructing truth-preserving liftings for fibrations other than codomain fibrations and the families fibration over Set. By contrast with the approach outlined in the conference paper [9] on which this paper is based, the one we take here uses a factorisation, like that of the previous subsection, through a codomain fibration. More specifically, let $U : \mathcal{E} \to \mathcal{B}$ be a comprehension category with unit. We first define functors I and π, and construct an adjunction $I \dashv \pi$ between $\mathcal{E}$ and $\mathcal{B}^{\rightarrow}$ such that the relationships indicated by the following diagram hold:

$$
\begin{tikzcd}
\mathcal{E} \arrow[rr, bend left, "\pi"] \arrow[dr, "U"'] & & \mathcal{B}^{\rightarrow} \arrow[ll, bend left, "I"] \arrow[ll, phantom, "\top"] \arrow[dl, "cod"] \\
& \mathcal{B} &
\end{tikzcd}
$$

We then use the adjunction indicated in the diagram to construct truth-preserving a lifting for U from that for the codomain fibration over $\mathcal{B}$.

To define the functor $\pi : \mathcal{E} \to \mathcal{B}^{\rightarrow}$ we generalise the definition of $\pi : \mathcal{P} \to \mathsf{Set}^{\rightarrow}$ from Sections 3 and 4.3.2. This requires us to work with the axiomatic characterisation in Definition 4.7 of the comprehension functor $\{-\} : \mathcal{E} \to \mathcal{B}$ as the right adjoint to the truth functor $K_1 : \mathcal{B} \to \mathcal{E}$. The counit of the adjunction $K_1 \dashv \{-\}$ is a natural transformation $\epsilon : K_1\{-\} \to Id$. Applying U to ϵ gives the natural transformation $U\epsilon : UK_1\{-\} \to U$, but since $UK_1 = Id$, in fact we have that $U\epsilon : \{-\} \to U$. We can therefore define π to be $U\epsilon$. Then π is indeed a functor from $\mathcal{E}$ to $\mathcal{B}^{\rightarrow}$, its action on an object P is π_P, and its action on a morphism $(f, f^{\sim})$ is $(\{(f, f^{\sim})\}, f)$.

We next turn to the definition of the left adjoint I to π. To see how to generalise the inverse image construction to more general fibrations we first recall from Example 4 that, if $f : X \to Y$ is a function and $P : Y \to \mathsf{Set}$, then $f^*P = P \circ f$. We can extend this mapping to a reindexing functor $f^* : \mathcal{E}_Y \to \mathcal{E}_X$ by defining $f^*(id, h^\sim) = (id, h^\sim \circ f)$. If we define the action of $\Sigma_f : \mathcal{E}_X \to \mathcal{E}_Y$ on objects by

$$\Sigma_f P \;=\; \lambda y. \biguplus_{\{x \mid f\,x = y\}} P\,x$$

where $\biguplus$ denotes the disjoint union operator on sets, and its action on morphisms by taking $\Sigma_f\,(id, \alpha^\sim)$ to be $(id,\ \forall (y : Y).\ \lambda(x : X, p : fx = y, t : Px).\,(x, p, \alpha^\sim\,x\,t))$, then Σ_f is left adjoint to f^*. Moreover, if we compute

$$\Sigma_f\,(K_1\,X) \;=\; \lambda y. \biguplus_{\{x \mid f\,x = y\}} K_1 X x$$

and recall that, for any $x : X$, the set $K_1\,X\,x$ is a singleton, then $\Sigma_f\,(K_1 X)$ is clearly equivalent to the inverse image of f.

The above discussion suggests that, in order to generalise the inverse image construction to a more general fibration $U : \mathcal{E} \to \mathcal{B}$, we should require each reindexing functor f^* to have the opreindexing functor Σ_f as its left adjoint. As in [10], no Beck-Chevalley condition is required on these adjoints. The following result, which appears as Proposition 2.3 of [11], thus allows us to isolate the exact class of fibrations for which we will have sound generic induction rules.

Theorem 4.12. *A fibration $U : \mathcal{E} \to \mathcal{B}$ is a bifibration iff for every morphism f in $\mathcal{B}$ the reindexing functor f^* has left adjoint Σ_f.*

Definition 4.13. A *Lawvere category* is a bifibration which is also a comprehension category with unit.

We construct the left adjoint $I : \mathcal{B}^\to \to \mathcal{E}$ of π for any Lawvere category $U : \mathcal{E} \to \mathcal{B}$ as follows. If $f : X \to Y$ is an object of $\mathcal{B}^\to$, i.e., a morphism of $\mathcal{B}$, then we define $I\,f$ to be the object $\Sigma_f(K_1 X)$ of $\mathcal{E}$. To define the action of I on morphisms, let (α_1, α_2) be a morphism in $\mathcal{B}^\to$ from $f : X \to Y$ to $f' : X' \to Y'$ in $\mathcal{B}^\to$. Then (α_1, α_2) is a pair of morphisms in $\mathcal{B}$ such that the following diagram commutes:

$$\begin{tikzcd} X \arrow[r, "\alpha_1"] \arrow[d, "f"'] & X' \arrow[d, "f'"] \\ Y \arrow[r, "\alpha_2"'] & Y' \end{tikzcd}$$

We must construct a morphism from $\Sigma_f(K_1 X)$ to $\Sigma_{f'}(K_1 X')$ in $\mathcal{E}$. To do this, notice that $f'^{K_1 X'}_\S \circ K_1 \alpha_1 : K_1 X \to \Sigma_{f'}(K_1 X')$ is above $f' \circ \alpha_1$, and that it is also above $\alpha_2 \circ f$ since $f' \circ \alpha_1 = \alpha_2 \circ f$. We can then consider the morphism $f'^{K_1 X'}_\S \circ K_1 \alpha_1$ and use the universal property of the opcartesian morphism $f^{K_1 X}_\S$ to deduce the existence of a morphism $h : \Sigma_f(K_1 X) \to \Sigma_{f'}(K_1 X')$ above α_2. It is not difficult, using the uniqueness of the morphism h, to prove that setting this h to be the image of the morphism (α_1, α_2) makes I a functor. In fact, since $cod \circ \pi = U$, Result (i) on page 190 of [11] guarantees that, for any Lawvere category $U : \mathcal{E} \to \mathcal{B}$ the functor $I : \mathcal{B}^\to \to \mathcal{E}$ exists and is left adjoint to $\pi : \mathcal{E} \to \mathcal{B}^\to$.

We can now construct a truth-preserving lifting for any Lawvere category $U : \mathcal{E} \to \mathcal{B}$ and functor F on $\mathcal{B}$.

Theorem 4.14. *Let $U : \mathcal{E} \to \mathcal{B}$ be a Lawvere category and, for any functor F on $\mathcal{B}$, define the functor $\hat{F}$ on $\mathcal{E}$ by*

$$\begin{array}{rcl} \hat{F} & : & \mathcal{E} \to \mathcal{E} \\ \hat{F} & = & IF^{\to}\pi \end{array}$$

Then $\hat{F}$ is a truth-preserving lifting of F.

Proof. It is trivial to check that $\hat{F}$ is indeed a lifting. To prove that it is truth-preserving, we need to prove that $\hat{F}(K_1 X) \cong K_1(FX)$ for any functor F on $\mathcal{B}$ and object X of $\mathcal{B}$. We do this by showing that each of π, $F^{\to}$, and I preserves fibred terminal objects, i.e., preserves the terminal objects of each fibre of the total category which is its domain. Then since $K_1 X$ is a terminal object in the fibre $\mathcal{E}_X$, we will have that $\hat{F}(K_1 X) = I(F^{\to}(\pi(K_1 X)))$ is a terminal object in $\mathcal{E}_{FX}$, i.e., that $\hat{F}(K_1 X) \cong K_1(FX)$ as desired.

We first show that π preserves fibred terminal objects. We must show that, for any object X of $\mathcal{B}$, $\pi_{K_1 X}$ is a terminal object of the fibre of $\mathcal{B}^{\to}$ over X, i.e., is an isomorphism with codomain X. We prove this by observing that, if $\eta : Id \to \{-\}K_1$ is the unit of the adjunction $K_1 \dashv \{-\}$, then $\pi_{K_1 X}$ is an isomorphism with inverse η_X. Indeed, if ϵ is the counit of the same adjunction, then the facts that $UK_1 = Id$ and that K_1 is full and faithful ensure that $K_1 \eta_X$ is an isomorphism with inverse $\epsilon_{K_1 X}$. Thus, $\epsilon_{K_1 X}$ is an isomorphism with inverse $K_1 \eta_X$, and so $\pi_{K_1 X} = U\epsilon_{K_1 X}$ is an isomorphism with inverse $UK_1 \eta_X$, i.e., with inverse η_X. Since $K_1 X$ is a terminal object in $\mathcal{E}_X$ and $\pi_{K_1 X}$ is a terminal object in the fibre of $\mathcal{B}^{\to}$ over X, we have that π preserves fibred terminal objects.

It is not hard to see that $F^{\to}$ preserves fibred terminal objects: applying the functor F to an isomorphism with codomain X — i.e., to a terminal object in the fibre of $\mathcal{B}^{\to}$ over X — gives an isomorphism with codomain FX — i.e., a terminal object in the fibre of $\mathcal{B}^{\to}$ over FX.

Finally, if $f : X \to Y$ is an isomorphism in $\mathcal{B}$, then Σ_f is not only left adjoint to f^*, but also right adjoint to it. Since right adjoints preserve terminal objects, and since $K_1 X$ is a terminal object of $\mathcal{E}_X$, we have that $If = \Sigma_f(K_1 X)$ is a terminal object of $\mathcal{E}_Y$. Thus I preserves fibred terminal objects. □

We stress that, to define our lifting, the codomain functor over the base $\mathcal{B}$ of the Lawvere category need not be a fibration. In particular, $\mathcal{B}$ need not have pullbacks; indeed, all that is needed to construct our generic truth-preserving lifting $\hat{F}$ for a functor F on $\mathcal{B}$ is the existence of the functors I and π (and $F^{\to}$, which always exists). We nevertheless present the lifting $\hat{F}$ as the composition of π, $F^{\to}$, and I because this presentation shows it can be factored through $F^{\to}$. This helps motivate our definition of $\hat{F}$, thereby revealing parallels between it and $F^{\to}$ that would otherwise not be apparent. At the same time it trades the direct, brute-force presentation of $\hat{F}$ from [9] for an elegant modularly structured one which makes good use, in a different setting, of general results about comprehension categories due to Jacobs [11].

We now have the promised sound generic fibrational induction rule for every inductive functor F on the base of a Lawvere category. To demonstrate the flexibility of this rule, we now derive an induction rule for a data type and properties on it that cannot be modelled in Set. Being able to derive induction rules for fixed points of functors in categories other than Set is a key motivation for working in a general fibrational setting.

Example 7. The fixed point $Hyp = \mu F$ of the functor $FX = (X \to Int) \to Int$ is the data type of hyperfunctions. Since F has no fixed point in Set, we interpret it in the category

$\omega CPO_\perp$ of ω-cpos with $\perp$ and strict continuous monotone functions. In this setting, a property of an object X of $\omega CPO_\perp$ is an admissible sub-$\omega CPO_\perp$ P of X. Admissibility means that the bottom element of X is in P and P is closed under least upper bounds of ω-chains in X. This structure forms a Lawvere category [11, 12]; in particular, it is routine to verify the existence of its opreindexing functor. In particular, $\Sigma_f P$ is constructed for a continuous map $f : X \to Y$ and an admisible predicate $P \subseteq X$, as the intersection of all admissible $Q \subseteq Y$ with $P \subseteq f^{-1}(Q)$. The truth functor maps X to X, and comprehension maps a sub-$\omega CPO_\perp$ P of X to P. The lifting $\hat{F}$ maps a sub-$\omega CPO_\perp$ P of X to the least admissible predicate on FX containing the image of FP. Finally, the derived induction rule states that if P is an admissible sub-$\omega CPO_\perp$ of Hyp, and if $\hat{F}(P) \subseteq P$, then $P = Hyp$.

4.4. **An Algebra of Lifting.** We have proved that in any Lawvere category $U : \mathcal{E} \to \mathcal{B}$, any functor F on $\mathcal{B}$ has a lifting $\hat{F}$ on $\mathcal{E}$ which is truth-preserving, and thus has the following associated sound generic fibrational induction rule:

$$\begin{array}{lcl} \mathit{genfibind} & : & \forall\, (F : \mathcal{B} \to \mathcal{B})\ (P : \mathcal{E}).\ (\hat{F}\, P \to P) \to (\mu F \to \{P\}) \\ \mathit{genfibind}\, F\, P & = & \mathit{fold} \circ \Psi \end{array}$$

In this final subsection of the paper, we ask what kinds of algebraic properties the lifting operation has. Our first result concerns the lifting of constant functors.

Lemma 4.15. *Let $U : \mathcal{E} \to \mathcal{B}$ be a Lawvere category and let X be an object of $\mathcal{B}$. If F_X is the constantly X-valued functor on $\mathcal{B}$, then $\widehat{F_X}$ is isomorphic to the constantly K_1X-valued functor on $\mathcal{E}$.*

Proof. For any object P of $\mathcal{E}$ we have

$$\widehat{F_X} P = (I(F_X)^{\to}\pi)P = I(F_X\pi_P) = \Sigma_{F_X\pi_P} K_1 F_X\{P\} = \Sigma_{id} K_1 X \cong K_1 X$$

The last isomorphism holds because $id^* \cong Id$ and $\Sigma_{id} \dashv id^*$. □

Our next result concerns the lifting of the identity functor. It requires a little additional structure on the Lawvere category of interest.

Definition 4.16. A *full Lawvere category* is a Lawvere category $U : \mathcal{E} \to \mathcal{B}$ such that $\pi : \mathcal{E} \to \mathcal{B}^{\to}$ is full and faithful.

Lemma 4.17. *In any full Lawvere category, $\widehat{Id} \cong Id$*

Proof. By the discussion following Definition 4.13, $I \dashv \pi$. Since π is full and faithful, the counit of this adjunction is an isomorphism, and so $I\pi_P \cong P$ for all P in $\mathcal{E}$. We therefore have that

$$P \cong I\pi_P = \Sigma_{\pi_p} K_1\{P\} = \Sigma_{Id\,\pi_p} K_1(Id\,\{P\}) = (I\,Id^{\to}\,\pi)P = \widehat{Id}\,P$$

i.e., that $\widehat{Id}\,P \cong P$ for all P in $\mathcal{E}$. Because these isomorphisms are clearly natural, we therefore have that $\widehat{Id} \cong Id$. □

We now show that the lifting of a coproduct of functors is the coproduct of the liftings.

Lemma 4.18. *Let $U : \mathcal{E} \to \mathcal{B}$ be a Lawvere category and let F and G be functors on $\mathcal{B}$. Then $\widehat{F+G} \cong \hat{F} + \hat{G}$.*

Proof. We have

$$(\widehat{F+G})P = I((F+G)^{\rightarrow}\pi_P) = I(F^{\rightarrow}\pi_P + G^{\rightarrow}\pi_P) \cong I(F^{\rightarrow}\pi_P) + I(G^{\rightarrow}\pi_P) = \hat{F}P + \hat{G}P$$

The third isomorphism holds because I is a left adjoint and so preserves coproducts. □

Note that the statement of Lemma 4.18 does not assert the existence of either of the two coproducts mentioned, but rather that, whenever both do exist, they must be equal. Note also that the lemma generalises to any colimit of functors. Unfortunately, no result analogous to Lemma 4.18 can yet be stated for products.

Our final result considers whether or not there is anything fundamentally special about the lifting we have constructed. It is clearly the "right" lifting in some sense because it gives the expected induction rules. But other truth-preserving liftings might also exist and, if this is the case, then we might hope our lifting satisfies some universal property. In fact, under a further condition, which is also satisfied by all of the liftings of Hermida and Jacobs, and which we therefore regard as reasonable, we can show that our lifting is the only truth-preserving lifting. Our proof uses a line of reasoning which appears in Remark 2.13 in [10].

Lemma 4.19. *Let $U : \mathcal{E} \to \mathcal{B}$ be a full Lawvere category and let $\Box F$ be a truth-preserving lifting of a functor F on $\mathcal{B}$. If $\Box F$ preserves Σ-types — i.e., if $(\Box F)(\Sigma_f P) \cong \Sigma_{Ff}(\Box F)P$ — then $\Box F \cong \hat{F}$.*

Proof. We have

$$\begin{array}{rcl}
(\Box F)P & \cong & (\Box F)(\widehat{Id}P) \\
& \cong & (\Box F)(\Sigma_{\pi_P} K_1\{P\}) \\
& \cong & \Sigma_{F\pi_P}(\Box F)K_1\{P\} \\
& \cong & \Sigma_{F\pi_P} K_1 F\{P\} \\
& = & \hat{F}P
\end{array}$$

□

Finally, we can return to the question of the relationship between the liftings of polynomial functors given by Hermida and Jacobs and the liftings derived by our methods. We have seen that for constant functors, the identity functor, and coproducts of functors our constructions agree. Moreover, since Hermida and Jacobs' liftings all preserve Σ-types, Lemma 4.19 guarantees that in a full Lawvere category their lifting for products also coincides with ours.

5. Conclusion and future work

We have given a sound induction rule that can be used to prove properties of data structures of inductive types. Like Hermida and Jacobs, we give a fibrational account of induction, but we derive, under slightly different assumptions on fibrations, a generic induction rule that can be instantiated to *any* inductive type rather than just to polynomial ones. This rule is based on initial algebra semantics of data types, and is parameterised over both the data types and the properties involved. It is also principled, expressive, and correct. Our derivation yields the same induction rules as Hermida and Jacobs' when specialised to polynomial functors in the families fibration over Set and in other fibrations, but it also gives induction rules for non-polynomial data types such as rose trees, and for data types such as finite hereditary sets and hyperfunctions, for which no fibrational induction rules have previously been known to exist.

There are several directions for future work. The most immediate is to instantiate our theory to give induction rules for more sophisticated data types, such as nested types. These are exemplified by the data type of perfect trees given in Haskell-like syntax as follows:

$$\begin{array}{l} \mathit{data}\ \mathit{PTree}\ a : \mathsf{Set}\ \mathit{where} \\ \quad \mathit{PLeaf} : a \to \mathit{PTree}\ a \\ \quad \mathit{PNode} : \mathit{PTree}\ (a, a) \to \mathit{PTree}\ a \end{array}$$

Nested types arise as least fixed points of rank-2 functors; for example, the type of perfect trees is μH for the functor H given by $HF = \lambda X.\ X + F(X \times X)$. An appropriate fibration for induction rules for nested types thus takes $\mathcal{B}$ to be the category of functors on Set, $\mathcal{E}$ to be the category of functors from Set to $\mathcal{P}$, and U to be postcomposition with the forgetful functor from Section 3. A lifting $\hat{H}$ of H is given by $\hat{H}\,P\,X\,(\mathit{inl}\,a) = 1$ and $\hat{H}\,P\,X\,(\mathit{inr}\,n) = P\,(X \times X)\,n$. Taking the premise to be an $\hat{H}$-algebra gives the following induction rule for perfect trees:

$$\begin{array}{l} \mathit{indPTree} : \forall\ (P : \mathsf{Set} \to \mathcal{P}). \\ \qquad\qquad (UP = \mathit{PTree}) \to (\forall (X : \mathsf{Set})(x : X).\ P\,(\mathit{PLeaf}\ x)) \to \\ \qquad\qquad (\forall (X : \mathsf{Set})(t : \mathit{PTree}\,(X \times X).\,P\,(X \times X)\,t \to P\,(\mathit{PNode}\,t))) \to \\ \qquad\qquad\quad \forall (X : \mathsf{Set})(t : \mathit{PTree}\,X).\,P\,X\,t \end{array}$$

This rule can be used to show, for example, that $PTree$ is a functor.

Extending the above instantiation for the codomain fibration to so-called "truly nested types" [15] and fibrations is current work. We expect to be able to instantiate our theory for truly nested types, GADTs, indexed containers, dependent types, and inductive recursive types, but initial investigations show care is needed. We must ascertain which fibrations can model predicates on such types, since the codomain fibration may not give useful induction rules, as well as how to translate the rules to which these fibrations give rise to an intensional setting.

Matthes [15] gives induction rules for nested types (including truly nested ones) in an intensional type theory. He handles only rank-2 functors that underlie nested types (while we handle any functor of any rank with an initial algebra), but his insights may help guide choices of fibrations for truly nested types. These may in turn inform choices for GADTs, indexed containers, and dependent types.

Induction rules can automatically be generated in many type theories. Within the Calculus of Constructions [4] an induction rule for a data type can be generated solely from the inductive structure of that type. Such generation is also a key idea in the Coq proof assistant [5]. As far as we know, generation can currently be done only for syntactic classes of functors rather than for all inductive functors with initial algebras. In some type theories induction schemes are added as axioms rather than generated. For example, attempts to generate induction schemes based on Church encodings in the Calculus of Constructions proved unsuccessful and so initiality was added to the system, thus giving the

Calculus of Inductive Constructions. Whereas Matthes' work is based on concepts such as impredicativity and induction recursion rather than initial algebras, ours reduces induction to initiality, and may therefore help lay the groundwork for extending implementations of induction to more sophisticated data types.

Acknowledgement

We thank Robert Atkey, Pierre-Evariste Dagand, Peter Hancock, and Conor McBride for many fruitful discussions.

References

[1] T. Altenkirch and P. Morris. Indexed Containers. Proceedings, Logic in Computer Science, pp. 277–285, 2009.

[2] R. S. Bird and O. De Moor. *Algebra of Programming.* Prentice Hall, 1997.

[3] R. Bird and L. Meertens. Nested Datatypes. *Proceedings, Mathematics of Program Construction*, pp. 52–67, 1998.

[4] T. Coquand and G. Huet. The Calculus of Constructions. *Information and Computation* 76 (2-3), pp. 95–120, 1988.

[5] The Coq Proof Assistant. Available at `coq.inria.fr`

[6] P. Dybjer. A General Formulation of Simultaneous Inductive-Recursive Definitions in Type Theory. *Journal of Symbolic Logic* 65 (2), pp. 525–549, 2000

[7] N. Ghani, M. Abbott, and T. Altenkirch. Containers - Constructing Strictly Positive Types. *Theoretical Computer Science* 341 (1), pp. 3–27, 2005.

[8] N. Ghani and P. Johann. Foundations for Structured Programming with GADTs. *Proceedings, Principles of Programming Languages*, pp. 297–308, 2008.

[9] N. Ghani and P. Johann and C. Fumex. Fibrational Induction Rules for Initial Algebras. *Proceedings, Computer Science Logic*, pp. 336–350, 2010.

[10] C. Hermida and B. Jacobs. Structural Induction and Coinduction in a Fibrational Setting. *Information and Computation* 145 (2), pp. 107–152, 1998.

[11] B. Jacobs. Comprehension Categories and the Semantics of Type Dependency. *Theoretical Computer Science* 107, pp. 169–207, 1993.

[12] B. Jacobs. *Categorical Logic and Type Theory.* North Holland, 1999.

[13] P. Johann and N. Ghani. Initial Algebra Semantics is Enough! *Proceedings, Typed Lambda Calculus and Applications*, pp. 207–222, 2007.

[14] F. W. Lawvere. Equality in Hyperdoctrines and Comprehension Scheme as an Adjoint Functor. *Applications of Categorical Algebra*, pp. 1–14, 1970.

[15] R. Matthes. An Induction Principle for Nested Datatypes in Intensional Type Theory. *Journal of Functional Programming* 19 (3&4), pp. 439–468, 2009.

[16] N. P. Mendler. Predicative type universes and primitive recursion. *Proceedings, Logic in Computer Science*, pp. 173–184, 1991.

[17] P. Morris. *Constructing Universes for Generic Programming.* Dissertation, University of Nottingham, 2007.

[18] E. Moggi. Notations of Computation and Monads. *Information and Computation* 93 (1), pp. 55–92, 1991.

[19] B. Nordström, K. Petersson, and J. Smith. *Programming in Martin-Löf's Type Theory.* Oxford University Press, 1990.

[20] D. Pavlovič. *Predicates and Fibrations.* Dissertation, University of Utrecht, 1990.

[21] T. Sheard. Languages of the Future. *SIGPLAN Notices* 39 (10), pp. 116–119, 2004.