Fibred Data Types

Neil Ghani, Lorenzo Malatesta
The MSP Group,
University of Strathclyde.
Email: {ng, lorenzo.malatesta}@cis.strath.ac.uk

Fredrik Nordvall Forsberg, Anton Setzer
Dept. of Computer Science,
Swansea University.
Email: {csfnf, a.g.setzer}@swansea.ac.uk

***Abstract*—Data types are undergoing a major leap forward in their sophistication driven by a conjunction of i) theoretical advances in the foundations of data types; and ii) requirements of programmers for ever more control of the data structures they work with. In this paper we develop a theory of indexed data types where, crucially, the indices are generated inductively at the same time as the data. In order to avoid commitment to any specific notion of indexing we take an axiomatic approach to such data types using fibrations – thus giving us a theory of what we call fibred data types.**

The genesis of these fibred data types can be traced within the literature, most notably to Dybjer and Setzer's introduction of the concept of induction-recursion. This paper, while drawing heavily on their seminal work for inspiration, gives a categorical reformulation of Dybjer and Setzer's original work which leads to a large number of extensions of induction-recursion. Concretely, the paper provides i) conceptual clarity as to what induction-recursion fundamentally is about; ii) greater expressiveness in allowing not just the inductive-recursive definition of families of sets, or even indexed families of sets, but rather the inductive-recursive definition of a whole host of other structures; iii) a semantics for induction-recursion based not on the specific model of families, but rather an axiomatic model based upon fibrations which therefore encompasses diverse structures (domain theoretic, realisability, games etc) arising in the semantics of programming languages; and iv) technical justification as to why these fibred data types exist using large cardinals from set theory.

I. Introduction

An inductive type is, informally, the least set closed under certain operations. These operations can be thought of as building elements of the data type and hence are called *constructors*. The fact that an inductive type is *the least* set closed under such operations means that the data type possesses a recursor, or elimination rule, which embeds the data type within any other set with enough structure to model the constructors. This intuition can be seen in, for example, the following Agda data type of lists storing data of type A

```
data List (A : Set) : Set where
  nil  : List A
  cons : A → List A → List A

fold : {A B : Set} →  B → (A → B → B)
       → List A → B
fold n c nil = n
fold n c (cons x xs) = c x (fold n c xs)
```

which has two constructors nil and cons which construct, respectively, the empty list and a list made from adding an element to the front of a list. The arguments for the recursor fold can be thought of as replacement operations n and c for nil and cons – thus list built from nil is mapped by fold n c to n while a list built from cons is mapped by fold n c to the result of applying c to the head of the list and the result of the recursive call.

The trouble with informal intuitions such as given above is that they are not robust enough to cleanly scale to more complex situations. Such situations arise, for example, if our data types i) are required to support more programming language structure such as induction principles, impredicative encodings, short cut fusion rules etc; ii) are not to be interpreted as sets but other mathematical objects such as domains, PERs, games etc; and iii) are more than simple tree like structures defined in a sums-and-products fashion. A more formal and robust approach to understanding data types is given by initial algebra semantics where inductive types are thought of as arising as the initial algebra of a functor $F : \mathcal{C} \to \mathcal{C}$ where $\mathcal{C}$ is the ambient category containing our intended interpretation of types. We revisit the basics of initial algebra semantics in Section II, but recall for now that the data type List A arises as the initial algebra of the functor $FX = \mathbf{1} + A \times X$. Other data types such as the natural numbers, trees etc. arise in a similar fashion. Typically, not all functors have initial algebras and so an important part of any successful theory of data types are grammars for generating functors which do indeed have initial algebras under mild assumptions. Examples of grammars for generating simple data types such as natural numbers, lists and trees etc., include the polynomial data types, strictly positive types and containers [2].

Unfortunately, inductive types such as lists and trees are too simple to capture important data types. As a result, one of the key aims of current research in functional programming is to reduce the semantic gap between what programmers know about computational entities and what the types of those entities can express about them. One particularly promising approach to this problem is to consider data types indexed by extra information that can be used to express properties of data having those types. Canonical examples of such data types are typed and untyped λ-terms, red-black trees and balanced binary trees. The crucial point about these data types is that one cannot inductively define the data with a given index independently of data with another index. To illustrate this point, notice how the type List A above can be defined independently of any type List B for B distinct from A. Thus, the type constructor List

defines a family of inductive types. Compare the declaration for List A with the declaration

```
data Lam (n : ℕ) : Set where
  var : Fin n → Lam n
  app : Lam n → Lam n → Lam n
  abs : Lam (suc n) → Lam n
```

defining the data type Lam n of untyped λ-terms over n free variables up to α-equivalence. Here Fin n is a type with n-elements. By contrast with List A, the type Lam n cannot be defined in terms of only those elements of Lam n that have already been constructed. Indeed, elements of the type Lam (suc n) are needed to build elements of Lam n so that, in effect, the entire family of types determined by Lam has to be constructed simultaneously. Thus, rather than defining a family of individual inductive types as List does, Lam defines an inductive family of types indexed by the natural numbers.

Fortunately, initial algebra semantics is robust enough to explain these inductive families. That is, one can simply apply the standard theory of initial algebra semantics in a more sophisticated category so as to derive recursion operators, induction principles, Church encodings and short cut fusion rules etc. for such inductive families. In the example of untyped λ-terms, one could choose the category $[\mathbb{N}, \mathsf{Sets}]$ of $\mathbb{N}$-indexed sets. The data type Lam then arises as the initial algebra of the functor $F_{\mathsf{Lam}} : (\mathbb{N} \to \mathsf{Sets}) \to \mathbb{N} \to \mathsf{Sets}$ defined by

$$F_{\mathsf{Lam}}\ X\ n = \mathsf{Fin}\ n\ +\ (X\ n) \times (X\ n)\ +\ X(n+1)$$

As in the unindexed setting, grammars have been proposed for defining well behaved functors on such indexed sets. Examples are the syntactic formats for inductive families of Dybjer [12], dependent polynomials of Gambino and Hyland [16] and the indexed containers of Altenkirch and Morris [5].

The example of the inductive family Lam is typical of the state-of-the art in data type design in that the indexes – $\mathbb{N}$ in this case – are defined in advance of the data type declaration – in this case Lam. But what if they are not? What if, as we build up the data, we also build up the indexes? Very important data types arise in this manner, for example the construction of universes/meta-languages for dependent types. To understand this, first recall that a universe is a pair (U, T) where we think of the indexing set $U : \mathsf{Sets}$ as consisting of the names for types and the decoding function $T : U \to \mathsf{Sets}$ as assigning to each name $u : U$ the type Tu of elements of the type named by u. Now say that we want to define a data type representing the smallest universe containing a name for the natural numbers and closed under Σ-types/dependent sums. Such a universe will be the least solution of the equations

$$\begin{array}{lcl} U & = & \mathbf{1} + \Sigma u{:}U.\ Tu \to U \\ T(\mathsf{inl}\ *) & = & \mathbb{N} \\ T(\mathsf{inr}\ (u, f)) & = & \Sigma x{:}Tu.\ T(fx) \end{array}$$

where the Σ on the right hand side of the above equations is the usual dependent sum. To understand the above equations, note that the dependent sum $\Sigma\, A\, B$ consists of a type A and a function B mapping each element of A to a type. Thus the name of a Σ-type in a universe (U, T) will consist of a name in U for the type A, i.e. an element $u : U$, and a function $f : Tu \to U$ assigning to every element of the type denoted by u, i.e. every element of Tu, the name of a type, i.e. an element of U. An element of the type denoted by a name (u, f) consists of an element of the type denoted by u, i.e. an element $x : Tu$, and an element of the type whose name is fx, i.e. an element of $T(fx)$. This example demonstrates a key point – the set U cannot be defined in advance of the function $T : U \to \mathsf{Sets}$. Such a definition is therefore not an example of an inductive family. As we shall see, it is a simple example of a fibred data type.

Our central ambition in this paper is to put such fibred data types on as sound a footing as are the simpler inductive types and inductive families. Our paper builds on the seminal work of Dybjer and Setzer's theory of induction-recursion [14] and we extend their work in a number of ways:

- We believe induction-recursion has not caught on as it should have and part of the problem is that its conceptual foundations need clarification. We demonstrate how greater conceptual clarity can be achieved by viewing induction-recursion from a fibrational perspective.
- Dybjer and Setzer's work allows the inductive-recursive definition of families and indexed families. We show how our fibrational perspective extends this to allow the inductive-recursive definition of a whole host of other structures.
- Dybjer and Setzer gave a semantics for induction-recursion based upon a specific model based upon the families fibration. We provide an axiomatic semantics based not on families, but rather on fibrations with certain structure.
- We generalise Dybjer and Setzer's soundness proof to show that fibred data types really do exist, i.e. that we can build initial algebras of the functors we consider.

This paper is structured as follows: In Section II we recap initial algebra semantics of data types. In Section III, we first present Dybjer and Setzer's induction-recursion, then show how it can be generalised to the fibrational setting, and finally show that all of the data types definable by our approach really do exist, that is, all of the functors we define have initial algebras. We conclude in Section IV with ideas for future work. Our methodology in this paper is inherently semantic – we look for mathematical constructions which we can then reflect back into the syntax of programming languages. Nevertheless, we sometimes take the perspective that syntax comes first and we then, subsequently, look for semantics for the syntax. Both perspectives have value and hence we do not feel the need to stick to only one.

II. Initial Algebra Semantics

Initial algebra semantics is one of the cornerstones of the theory of modern functional programming languages. Therefore, and in order to both fix our notion and to provide a backdrop for this paper, we give a brief summary of how initial algebra semantics can be used on the one hand to provide a semantics to data

types and, on the other hand, to offer a mathematical theory of computation which can be reflected back into programming language constructs.

Within the paradigm of initial algebra semantics, each data type is regarded as the carrier of the initial algebra of a functor F. In more detail, if $\mathcal{B}$ is a category and F is a functor on $\mathcal{B}$, then an *F-algebra* is a pair (X, h) where X is an object of $\mathcal{B}$ and $h : FX \to X$. We call X the *carrier* of the algebra.

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 homomorphism* from (X, h) to (Y, g) is a map $f : X \to Y$ such that the following diagram commutes

$$\begin{array}{ccc} FX & \xrightarrow{h} & X \\ {\scriptstyle Ff}\downarrow & & \downarrow{\scriptstyle f} \\ FY & \xrightarrow[g]{} & Y \end{array}$$

The initial F-algebra $(\mu F, in_F)$ is the initial object in this category. When it exists it is unique up to isomorphism. The object μF is the interpretation of the data type described by F, while the morphism $in_F : F(\mu F) \to \mu F$ interprets its constructors. Initiality ensures that, given any F-algebra $h : FX \to X$, there is a unique F-algebra homomorphism, denoted $fold\ h$ from the initial algebra $(\mu F, in_F)$ to that algebra. For each functor F, this defines a recursion operator $fold : (FX \to X) \to \mu F \to X$ which is the semantic counterpart of the elimination rule for the data type μF. The fact that $fold\ h$ is actually an algebra homomorphism corresponds to β-equality for the data type μF, while the fact that $fold\ h$ is unique models η-equality.

Of course, in most categories, not all functors have initial algebras and so this leads us to augment initial algebra semantics with specific grammars for defining functors which have initial algebras. For example, inductive types like $\mathbb{N}$ and $\mathsf{List}\ A$ can be described by containers [2].

Example 1. A container is a pair (S, P) consisting of a set $S : \mathsf{Sets}$ of *shapes* and a function $P : S \to \mathsf{Sets}$ assigning to each shape $s : S$, a set of *positions* Ps. Intuitively, one may think of S as a set of operators and P as assigning to each operator its arity. Every container (S, P) defines a functor $[\![S, P]\!] : \mathsf{Sets} \to \mathsf{Sets}$ as follows

$$[\![S, P]\!]\, X = \Sigma s{:}S.\ \ Ps \to X$$

For example, the inductive type $\mathbb{N}$ arises as the initial algebra of the container with two shapes $S = \{z, s\}$ and positions $P(z) = \emptyset$, $P(s) = \{\star\}$, since it has two operations $\mathsf{0} : \mathbb{N}$ and $\mathsf{suc} : \mathbb{N}{\to}\mathbb{N}$ respectively of arity zero and one.

In fact the theory of containers can be developed in any locally Cartesian closed category. The shapes and positions metaphor underlying containers means that they possess a very rich algebraic structure [1].

At the next level of sophistication, we want to capture indexed data types within the initial algebra paradigm. A common example of indexed data types are the *inductive families* where one first fixes a set U and then defines a U-indexed family of types $T : U \to \mathsf{Sets}$. Crucially, U here is a fixed set defined independently and in advance of the inductive family T. An example of such a data type is the type Lam given in the introduction where U is the set of natural numbers.

Many different grammars for functors whose initial algebras are inductive families can be found in the literature [12] [16] [9]. We mention here the indexed containers of Altenkirch and Morris [5].

Example 2. An (I, O)-indexed container is a pair (S, P) consisting of an indexed set $S{:}O \to \mathsf{Sets}$ of *shapes* and a function $P : (o : O) \to S\, o \to I \to \mathsf{Sets}$ assigning to each shape $s : S\, o$, an I-indexed set of *positions*. Every (I, O)-indexed container (S, P) defines a functor $[\![S, P]\!] : (I \to \mathsf{Sets}) \to O \to \mathsf{Sets}$ as follows

$$[\![S, P]\!]\, X\, o = \Sigma s{:}S\, o.\ \ \Pi i : I.\, P\, o\, s\, i \to X\, i$$

Note how the input and output index sets I and O are fixed in advance.

To summarise, initial algebra semantics provides a well-developed theory of data types which is i) principled, and so helps to ensure that programs have precise mathematical foundations that can be used to ascertain/predict their meaning and correctness; ii) expressive, and so is applicable to all data types, rather than just syntactically defined classes of data types such as polynomial data types etc; and iii) axiomatic, and so is valid in any model – set-theoretic, domain-theoretic, game-theoretic, realisability, etc. – in which data types are interpreted as carriers of initial algebras. Its success can be further measured by the fact that it is used as a framework both by researchers in the mathematical foundations of programming language semantics, by programming language designers and even by programmers themselves (under the guise of the *Algebra of Programming School* [7]).

III. Fibred Induction-Recursion

We have seen that indexing is an essential feature of data type design allowing us to define data types which match our intuition about the properties we want our data types to capture. However, most current theories enable us to only define indexed data types whose indexes are defined in advance of the actual data. For example, in an inductive family, one has to first say what U is and, only then, does one define $T : U \to \mathsf{Sets}$. A slight variant on the example given in the introduction is the following: if we wish to define a universe of types containing the natural numbers and closed under Π-types, then we are naturally lead to consider solutions of equations of the form

$$\begin{array}{lcl} U & = & \mathbf{1} + \Sigma u{:}U.\ Tu \to U \\ T(\mathsf{inl}\ *) & = & \mathbb{N} \\ T(\mathsf{inr}\ (u, f)) & = & \Pi x{:}Tu.\ T(fx) \end{array}$$

where $U : \mathsf{Sets}$ and $T : U \to \mathsf{Sets}$. Notice in this example how the name for a Π-type in this universe is still a choice of a name for the type we quantify over and an assignment of another name to each element of that type – hence we still use

Σ when defining U. However, the decoding of such a name via T uses Π so that we do indeed decode such a name to a Π-type. We introduce this universe now as we shall soon see the subtle difference mathematically between universes closed under Σ-types and Π-types. But for now, note that once more U cannot be defined in advance of T. So, the question is, can we give a predictive foundation for such data types?

Actually, Dybjer and Setzer did this 10 years ago in their theory of *induction-recursion* [13]. Their key observation is this – if U is a collection of indexes not fixed in advance then the semantic category we work in must contain all functions $T : U \to \mathsf{Sets}$ for *all* sets U. Dybjer and Setzer do this by using the standard families construction from category theory:

Definition 1. Let $\mathcal{C}$ be a category. The category $\mathsf{Fam}(\mathcal{C})$ is defined as follows.

- The objects of $\mathsf{Fam}(\mathcal{C})$ are pairs $(U : \mathsf{Sets}, T : U \to |\mathcal{C}|)$. Here, $|\mathcal{C}|$ refers to the collection of objects of the category $\mathcal{C}$. Note here that U is a set. We call U the index set of the family and T the decoding function of the family.
- The morphisms of $\mathsf{Fam}(\mathcal{C})$ between (U, T) and (U', T') consist of pairs (f, g) where $f : U \to U'$ is a function between the index sets of the respective families and g is a function assigning to each $u : U$ a morphism $g_u : Tu \to T'(fu)$ in $\mathcal{C}$. Equivalently, g is a natural transformation between T and $T' \circ f$ regarded as functors $U \to \mathcal{C}$.

The object part of an endofunctor $F : \mathsf{Fam}(\mathcal{C}) \to \mathsf{Fam}(\mathcal{C})$ can be given in two parts: i) a function $F^0 : \mathsf{Fam}(\mathcal{C}) \to \mathsf{Sets}$ such that for any family $X : \mathsf{Fam}(\mathcal{C})$, $F^0 X$ is the index of the family FX; and ii) a function $F^1 : \Pi X : \mathsf{Fam}(\mathcal{C}) \,.\, F^0 X \to |\mathcal{C}|$ which assigns to each object X of $\mathsf{Fam}(\mathcal{C})$ a function $F^0 X \to |\mathcal{C}|$ which is the decoding function of the family FX.

One may hope that we can construct indexed data types where the index is generated at the same time as the data itself by taking initial algebras of such endofunctors on $\mathsf{Fam}(\mathcal{C})$. But there is a slight snag. While the equations defining the functor generating a universe closed under Σ-types do indeed define such a functor, the equations defining a universe closed under Π-types do not! The problem is that, in the case of a universe closed under Π-types, we cannot define the action of the functor on the morphisms of $\mathsf{Fam}(\mathsf{Sets})$. Perhaps, on reflection, this is not surprising – Π-types generalise function spaces and there is an inherent contravariance in function spaces.

Dybjer and Setzer solve this problem by working not with categories of the form $\mathsf{Fam}(\mathcal{C})$, but rather with categories of the form $\mathsf{Fam}(|\mathcal{C}|)$, which has the same objects as $\mathsf{Fam}(\mathcal{C})$, but whose morphisms from $T : U \to |\mathcal{C}|$ to $T' : U' \to |\mathcal{C}|$ are simply functions $f : U \to U'$ such that $T = T' \circ f$. More precisely, they consider categories of the form $\mathsf{Fam}(|D|)$, where $|D|$ is the discrete category induced by a type D: the objects of $|D|$ are the elements of D, and the only morphisms are identity morphisms.

Putting all this together, Dybjer and Setzer defined a system of codes for defining functors as follows:

Definition 2 (Dybjer and Setzer's IR-Codes)**.** Let D be a potentially large set. The large set $\mathsf{IR}\ D$ of IR-codes has the following constructors

$$\frac{d : D}{\iota\ d : \mathsf{IR}\ D}$$

$$\frac{A : \mathsf{Sets} \quad f : A \to \mathsf{IR}\ D}{\sigma_A f : \mathsf{IR}\ D}$$

$$\frac{A : \mathsf{Sets} \quad F : (A \to D) \to \mathsf{IR}\ D}{\delta_A F : \mathsf{IR}\ D}$$

Each IR-code in $\mathsf{IR}(D)$ can be interpreted as an endofunctor on $\mathsf{Fam}(|D|)$ as follows:

Definition 3 (Dybjer and Setzer's semantics)**.** Let D be a large set and $c : \mathsf{IR}\ D$. Define the object part of the functor $[\![c]\!] : \mathsf{Fam}(|D|) \to \mathsf{Fam}(|D|)$ as follows:

- When $c = \iota d$,

$$[\![\iota\ d]\!]^0\ (U, T) = \mathbf{1}$$
$$[\![\iota\ d]\!]^1\ (U, T)\ \_ = d$$

- When $c = \sigma_A f$,

$$[\![\sigma_A f]\!]^0\ (U, T) = \Sigma a : A.\ [\![fa]\!]^0\ (U, T)$$
$$[\![\sigma_A f]\!]^1\ (U, T)\ (a, i) = [\![fa]\!]^1\ (U, T)\ i$$

- When $c = \delta_A F$,

$$[\![\delta_A F]\!]^0\ (U, T) = \Sigma g : A \to U.\ [\![F(T \circ g)]\!]^0\ (U, T)$$
$$[\![\delta_A F]\!]^1\ (U, T)\ (g, i) = [\![F(T \circ g)]\!]^1\ (U, T)\ i$$

Dybjer and Setzer also define the morphism part of $[\![c]\!]$, which we omit here. It will be an instance of our construction in Lemma 4. In the definition of decoding notice that various coproducts in Sets are used. Crucially, these coproducts are small and hence exist. For example, the definition of $[\![\sigma_A f]\!]$ is a coproduct over elements of the small set A, while in the definition of $[\![\delta_A F]\!](U, T)$, the coproduct is over elements of $A \to U$. Since A and U are small sets, such elements also form a set.

So the main question is ... what does the above mean? It looks like fairly technical type theory and many researchers have tried, and found it hard, to understand IR-codes and their semantics in the form of their decoding function. This is clearly a problem since induction-recursion has enough potential that it deserves study from a variety of researchers with varying perspectives and backgrounds. We hope our algebraic perspective on induction-recursion, and in particular its presentation via familiar and known categorical concepts will help to rectify this situation. A more concrete motivation for this paper arises when we ask what happens if we wish to define some other form of indexed structure where the indexes are generated at the same time as the data so indexed. For example, we may wish to define any of the following which are not families and hence are not covered by Dybjer and Setzer's work.

- A set-valued relation $T : U \times U \to \mathsf{Sets}$.

- An extensional family $T : U \to \mathsf{Setoid}$ where U is a setoid and T preserves the setoid structure of U.
- A presheaf $T : \mathcal{C} \to \mathsf{Sets}$ where $\mathcal{C}$ is a category and T is a functor.
- A category with families which is a functor $T : \mathcal{C}^{\mathrm{op}} \to \mathsf{Fam}(\mathsf{Sets})$ where $\mathcal{C}$ is a category thought of as a category of contexts.
- An indexed category/split fibration $T : \mathcal{C}^{\mathrm{op}} \to \mathsf{Cat}$.

To provide more conceptual clarity about what induction-recursion is, and to extend induction-recursion to cover the above examples we need to generalise Dybjer and Setzer's theory of induction-recursion. Rather than choosing different forms of indexing on an ad-hoc basis covering each of the above examples in turn, we prefer to use an axiomatic approach to indexing which can be instantiated to any of the above examples and more. To do so, we turn to the algebraic axiomatisation of indexing as given by fibrations.

A. Fibrations in a Nutshell

We give a unifying axiomatic approach to induction-recursion based on fibrations motivated by the facts that i) we are interested in using induction-recursion to define a whole host of structures beyond simply universes $T : U \to \mathsf{Sets}$; ii) the semantics of data types in languages involving recursion and other effects usually involves categories other than Sets; iii) in such circumstances, decoding functions can no longer be taken to be just a function with codomain Sets; iv) even when using essentially set-theoretic reasoning we may wish to use the proof relevant notion of a setoid rather than a mere set; and v) when using induction-recursion for more sophisticated structures, we will not want to have to develop an individual theory of induction-recursion specifically for each such type of structure. Instead, we will want to obtain inductive-recursive definitions by appropriately instantiating a single, generic theory of induction-recursion. That is, we will want to instantiate a uniform, axiomatic approach to induction-recursion that is widely applicable, and that abstracts over the specific choices of index and the codomain of the decoding function. Fibrations [18] support precisely such an axiomatic approach and so we turn to them.

Definition 4. Let $K : \mathcal{E} \to \mathcal{B}$ be a functor. A morphism $g : Q \to P$ in $\mathcal{E}$ is *Cartesian* over a morphism $f : X \to Y$ in $\mathcal{B}$ if $K(g) = f$, and for every $g' : Q' \to P$ in $\mathcal{E}$ for which $K(g') = f \circ v$ for some $v : K\,Q' \to X$ there exists a unique $h : Q' \to Q$ in $\mathcal{E}$ such that $K(h) = v$ and $g \circ h = g'$.

It is not hard to see that a Cartesian morphism $f^{\S}_P$ over a morphism f with codomain $K\,P$ is unique up to isomorphism. If P is an object of $\mathcal{E}$, then we write f^*P for the domain of $f^{\S}_P$. Cartesian morphisms are the essence of fibrations, as the following definition shows.

Definition 5. Let $K : \mathcal{E} \to \mathcal{B}$ be a functor. Then K is a *fibration* if for every object P of $\mathcal{E}$ and every morphism $f : X \to KP$ in $\mathcal{B}$ there is a Cartesian morphism $f^{\S}_P : Q \to P$ in $\mathcal{E}$ such that $K(f^{\S}_P) = f$.

If $K : \mathcal{E} \to \mathcal{B}$ is a fibration, we call $\mathcal{B}$ the *base category* of K and $\mathcal{E}$ the *total category* of K. In the rest of the paper we assume the base category is locally small – this will enable us later to take coproducts indexed by the morphisms of specific homsets. Objects of the total category $\mathcal{E}$ can be thought of as indexed entities while objects of the base category $\mathcal{B}$ can be thought of as indexes, and K can be thought of as mapping each indexed structure P in $\mathcal{E}$ to the index KP of P. We say that an object P in $\mathcal{E}$ is *above* its image KP under K, 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_X. If $f : X \to Y$ is a morphism in $\mathcal{B}$, then the function mapping each object P of $\mathcal{E}$ to f^*P extends to a functor $f^* : \mathcal{E}_Y \to \mathcal{E}_X$. If we think of f as an index-level substitution, then f^* can be thought of as lifting f to act on indexed structures. We call the functor f^* the *reindexing functor induced by* f.

A fibration is called *cloven* if it comes with a choice of Cartesian liftings, and *split* if this choice is further done functorially, i.e. $\mathsf{id}^* = \mathsf{id}$ and $(v \circ u)^* = u^*\,v^*$. As is common when modelling type theories, we will only be concerned with split fibrations. This is not a restriction, since every fibration is equivalent to a split one (Jacobs [18] Corollary 5.2.5). When K is split, we can consider the subcategory $\mathcal{E}^{\mathsf{sp}}$ of $\mathcal{E}$ consisting of Cartesian morphisms arising from the splitting only. Such morphisms are called *splitting morphisms*.

Example 3. In Definition 1, we defined the category $\mathsf{Fam}(\mathcal{C})$ for any category $\mathcal{C}$. The functor $\mathsf{Fam}(\mathcal{C}) \to \mathsf{Sets}$ mapping (U, T) to U is a split fibration called the *families fibration*. Given a family $T : U \to |\mathcal{C}|$ above U and a morphism $f : U' \to U$ in Sets, the reindexing of T by f is the family $T \circ f : U' \to |\mathcal{C}|$ with the chosen Cartesian arrow from $T \circ f$ to T having as first component f and as second component the U'-indexed collection of identity morphisms.

That $\mathsf{Fam}(|\mathcal{C}|)$ – the model proposed for induction-recursion by Dybjer and Setzer – is fibred over Sets is the crucial observation which allowed us to realise that induction-recursion can be recast in a fibred setting. But before we get to that, we give some other examples of fibrations.

Example 4. Let $\mathcal{B}$ be a category. The *arrow category* of $\mathcal{B}$, denoted $\mathcal{B}^{\to}$, has the morphisms of $\mathcal{B}$ as its objects. A morphism in $\mathcal{B}^{\to}$ from $f : X \to Y$ to $f' : X' \to Y'$ is a pair (α_1, α_2) of morphisms in $\mathcal{B}$ such that $f' \circ \alpha_1 = \alpha_2 \circ f$. The domain functor dom maps an object $f : X \to Y$ of $\mathcal{B}^{\to}$ to the object X of $\mathcal{B}$. This functor is always a fibration – called (unsurprisingly) the *domain fibration* – with the reindexing of an object $f : X \to Y$ over X by a morphism $g : X' \to X$ in $\mathcal{B}$ being simply the composite $f \circ g : X' \to Y$. For each object A of $\mathcal{B}$ there is a corresponding domain fibration $\mathsf{dom}_A : \mathcal{B}/A \to \mathcal{B}$ which maps $f : X \to A$ to its domain X.

Example 5. The category Rel of set-valued relations has as objects pairs $(U, T : U \times U \to \mathsf{Sets})$ where U is a set. A morphism from (U, T) to (U', T') is a pair (f, g) where f is function $f : U \to U'$ and, for each pair $u, u' : U$, we

have a function $g_{u,u'} : T(u,u') \to T'(fu,fu')$. Note how as expected, this category of relations is nothing more than a binary version of the category of families. As a result, the functor sending a relation $T : U \times U \to \mathsf{Sets}$ to U is a fibration for much the same reasons that hold for the families fibration. In particular, the reindexing of a relation $T : U \times U \to \mathsf{Sets}$ over U, by a morphism $h : U' \to U$ is the relation $T \circ (h \times h) : U' \times U' \to \mathsf{Sets}$.

B. Fibred Induction-Recursion

We introduce fibrations as they offer a conceptual cleaner treatment of induction-recursion which, simultaneously, significantly broadens the class of data types which can be defined by induction-recursion. These properties arise because fibrations form an axiomatic notion of model for induction-recursion rather than being a specific model. To understand this we look once more at Dybjer and Setzer's three constructors for IR-codes and contemplate how we can generalise them to an arbitrary fibration.

- **ι-codes:** The first of Dybjer and Setzer's three constructors of inductive recursive definitions is ι whose effect is to define a constant functor returning a given family. Because families are objects of the total category of the families fibration, we can generalise the ι constructor to an arbitrary fibration $K : \mathcal{E} \to \mathcal{B}$ by the following rule

$$\frac{P : \mathcal{E}}{\iota\ P : \mathsf{IR}\ K}$$

  whose intent is that ιP is a code for the constantly P valued functor on the total category of the fibration K.
- **σ-codes:** The second of Dybjer and Setzer's three constructors of inductive recursive definitions is σ whose effect is to take set-indexed coproducts of functors given by codes. Such an A-indexed collection of codes can be given by a function $f : A \to \mathsf{IR}\ K$. Hence we can generalise the σ constructor to an arbitrary fibration $K : \mathcal{E} \to \mathcal{B}$ by the following rule

$$\frac{A : \mathsf{Sets} \quad f : A \to \mathsf{IR}\ K}{\sigma_A f : \mathsf{IR}\ K}$$

- **δ-codes:** The third of Dybjer and Setzer's three constructors of inductive recursive definitions is δ. The premise of this constructor is a function $(A \to D) \to \mathsf{IR}D$ for a fixed index A. Notice that $A \to D$ is just an object in the families fibration $\mathsf{Fam}(D)$ in the fibre above A. Notice also that it is key to Dybjer and Setzer's δ constructor that F is in fact a function. Thus we may abstract δ to an arbitrary fibration $K : \mathcal{E} \to \mathcal{B}$ as follows:

$$\frac{A : \mathcal{B} \quad F : |\mathcal{E}_A| \to \mathsf{IR}\ K}{\delta_A F : \mathsf{IR}\ K}$$

Of course we still have to show that these abstractions make sense and in particular that they do indeed define appropriate functors. Nevertheless, we think that, already, we see the fibrational framework as paying dividends in terms of cleaning up the syntax of IR-codes.

Now we turn to the semantic content of our fibred IR-codes. The natural first guess is that if $K : \mathcal{E} \to \mathcal{B}$ is a fibration and $c : \mathsf{IR}\ K$ is a fibred IR-code, then there should be a functor $[\![c]\!]$ on $\mathcal{E}$, the total category of the fibration K. This seems in accordance with our motivating example of Dybjer and Setzer's codes which, from our perspective, arise from the families fibration. Of course, σ constructs set-indexed coproducts of functors and so $\mathcal{E}$ needs to have set-indexed coproducts:

Lemma 1. *Let $K : \mathcal{E} \to \mathcal{B}$ be a split fibration with set-indexed coproducts. Every fibred IR-code $c : \mathsf{IR}\ K$ induces a mapping $[\![c]\!]$ on the objects of $\mathcal{E}$.*

Proof: We go through each of the constructors in turn. Let $Q : \mathcal{E}$ be an object of the total category $\mathcal{E}$.

- If P is an object of $\mathcal{E}$, then the code $\iota\ P$ defines the constantly P valued map on objects of $\mathcal{E}$. Formally,

$$[\![\iota\ P]\!]\ Q = P$$

- Let A be a set and $f : A \to \mathsf{IR}\ K$ assign to each element $a : A$ an IR-code $fa : \mathsf{IR}\ K$. For each $a : A$, we have by the induction hypothesis an object $[\![fa]\!]\ Q$ of $\mathcal{E}$. Since $\mathcal{E}$ has set-indexed coproducts, we can take the coproduct of these objects to define $[\![\sigma_A f]\!]\ Q$. Formally,

$$[\![\sigma_A f]\!]\ Q = \Sigma a{:}A.\ \ [\![fa]\!]\ Q$$

- Let $A : \mathcal{B}$ and $F : |\mathcal{E}_A| \to \mathsf{IR}\ K$ assign to each object P in the fibre above A an IR-code FP. We can consider the maps $g : A \to KQ$ in $\mathcal{B}$. For each such g, we can reindex Q by the reindexing functor g^* to get an object g^*Q in the fibre above A. We can then apply F to this object to get an IR-code and inductively compute the action of this IR-code on Q. Notice that since $\mathcal{B}$ is locally small, there is a set of such maps and so we can take the set-indexed coproduct over the choice of g and thus obtain the action of $[\![\delta_A F]\!]$ on Q. Formally,

$$[\![\delta_A F]\!]\ Q = \Sigma g{:}A \to KQ.\ \ [\![F(g^*Q)]\!]\ Q$$ ∎

So our generalisation of IR-codes to fibrations also extends to showing how these fibred IR-codes define a map on the objects of the total category of the fibration. We may hope that in fact every fibred IR-code defines a functor on the total category of the fibration. Unfortunately, this is not true in general. Just as with Dybjer and Setzer's codes, we too have to be careful with the inherent contravariance in some universes such as the universe closed under Π-types we described before.

From our fibrational perspective, Dybjer and Setzer solved this problem by not working with arbitrary morphisms in $\mathsf{Fam}(\mathsf{Sets})$, but only those whose second component was the identity – that is they worked with morphisms between families $T : U \to \mathsf{Sets}$ and $T' : U \to \mathsf{Sets}$ which are functions $f : U \to U'$ such that $T = T' \circ f$. The same problem they faced arises here – its simply not the case that if $K : \mathcal{E} \to \mathcal{B}$ is a fibration then every fibrational IR code $c : \mathsf{IR}\ K$ defines a functor $[\![c]\!] : \mathcal{E} \to \mathcal{E}$. In fact, the morphisms which Dybjer and Setzer's IR-codes act on are exactly the splitting morphisms of

the families fibration. So, the natural generalisation of Dybjer and Setzer's condition on morphisms between families that works in an arbitrary fibration is simply that $[\![c]\!]$ acts only on the splitting morphisms of the fibration K. Formally, given a fibration $K : \mathcal{E} \to \mathcal{B}$ we can obtain a new fibration $K^{\mathsf{c}} : \mathcal{E}^{\mathsf{c}} \hookrightarrow \mathcal{E} \to \mathcal{B}$– called the discrete fibration associated to K [18]. We collect two immediate but useful facts about $\mathcal{E}^{\mathsf{sp}}$ in the following lemma:

Lemma 2. *Let $K : \mathcal{E} \to \mathcal{B}$ be a split fibration.*

(i) If f is a splitting morphism, then $(Kf)^{\S} = f$.
(ii) Each fibre $\mathcal{E}^{\mathsf{sp}}_A$ is discrete. ∎

Our goal is to show that for a split fibration $K : \mathcal{E} \to \mathcal{B}$, each fibred IR-code $c : \mathsf{IR}K$ gives rise to a functor $[\![c]\!] : \mathcal{E}^{\mathsf{sp}} \to \mathcal{E}^{\mathsf{sp}}$. Since coproducts play an important role in the construction, we need that they interact nicely with the splitting morphisms. Recall that (chosen) coproducts in a category $\mathcal{C}$ give rise to a functor $\Sigma : \mathsf{Fam}(C) \to \mathcal{C}$ which sends a family of objects to their set-indexed coproduct, and a family morphism (f, g) where $f : A \to A'$ and $g_x : B(x) \to B'(f(x))$ to the cotuple $\Sigma(f, g) = [\mathsf{in}_{f(x)} \circ g_x]_{x:A} : \Sigma(A, B) \to \Sigma(A', B')$. We call $\Sigma(f, g)$ the generalised sum of g over f. If $f = \mathsf{id}$, we simply write $\Sigma a\!:\!A.\ g_a$ for the sum.

Lemma 3. *Let $K : \mathcal{E} \to \mathcal{B}$ be a split fibration. If $\mathcal{E}$ has set-indexed coproducts, and if a generalised sum of splitting morphisms is splitting, then every IR-code $c : \mathsf{IR}\ K$ induces a functor $[\![c]\!] : \mathcal{E}^{\mathsf{sp}} \to \mathcal{E}^{\mathsf{sp}}$.*

Proof: In Lemma 1, we saw how $[\![c]\!]$ maps objects of $\mathcal{E}$ to objects of $\mathcal{E}$. As the objects of $\mathcal{E}$ and $\mathcal{E}^{\mathsf{sp}}$ are the same, this defines the action of $[\![c]\!]$ on the objects of $\mathcal{E}^{\mathsf{sp}}$. So now let us define the action of $[\![c]\!]$ on a splitting morphism $h : Q \to Q'$ by looking at the three constructors for fibred IR-codes in turn.

- If $c = \iota\ P$, then $[\![\iota\ P]\!]$ was defined to be the constantly P valued map on objects of $\mathcal{E}$. Therefore we can define
$$[\![\iota\ P]\!]\ h = \mathsf{id}_P$$
- If $c = \sigma_A f$, then
$$[\![\sigma_A f]\!]\ Q = \Sigma a\!:\!A.\ \ [\![fa]\!]\ Q$$
We have for every $a : A$ a splitting morphism $[\![fa]\!]\ h : [\![fa]\!]\ Q \to [\![fa]\!]\ Q'$. by assumption, the sum of all these morphisms is splitting and hence we can define
$$[\![\sigma_A f]\!]\ h = \Sigma a\!:\!A.\ \ [\![fa]\!]\ h$$
- If $c = \delta_A F$, then
$$[\![\delta_A F]\!]\ Q = \Sigma g\!:\!A \to KQ.\ [\![F(g^*Q)]\!]\ Q$$
Composition with $K(h)$ gives a map $K(h) \circ - : (A \to KQ) \to (A \to KQ')$, and by the induction hypothesis, we have a family of splitting morphisms $[\![F(g^*Q)]\!]\ h : [\![F(g^*Q)]\!]\ Q \to [\![F(g^*Q)]\!]\ Q'$. But by Lemma 3, $Q = K(h)^*Q'$, hence
$$g^*Q = g^*\ (K(h)^*Q') = (K(h) \circ g)^*Q'$$
Thus, the codomain of the morphism $[\![F(g^*Q)]\!]\ h$ is in fact $[\![F((K(h) \circ g)^*Q')]\!]\ Q'$, which is exactly what we need to form the generalised sum over $K(h) \circ -$. Thus we can define
$$[\![\delta_A F]\!]\ h = \Sigma(K(h) \circ -, [\![F(-^*Q)]\!]\ h)$$ ∎

Remark 1. Inspecting the proof, we see that it would go through with any functor $S : \mathsf{Fam}(\mathcal{E}^{\mathsf{sp}}) \to \mathcal{E}^{\mathsf{sp}}$ instead of Σ, but we expect coproducts to most closely model the data types we are interested in. Note that if the subcategory $\mathcal{E}^{\mathsf{sp}}$ is closed under coproducts, then the condition is automatically true.

We can also offer the following alternative proof, based on an "impredicative" decoding of a δ code. We make use of the following lemma, used to switch between global structure in the total category and local structure in the fibre.

Lemma 4 ([18] 1.4.10)**.** *Let $K : \mathcal{E} \to \mathcal{B}$ be a cloven fibration. There is a natural isomorphism*
$$\mathcal{E}(X, Y) \cong \Sigma g : K\,X \to K\,Y.\, \mathcal{E}_{KX}(X, g^*(Y))$$ ∎

If the fibration K is split, by Lemma 3 all fibres of K^{sp} are discrete and we can rephrase the isomorphism from Lemma 5 for $K^{\mathsf{sp}} : \mathcal{E}^{\mathsf{sp}} \to \mathcal{B}$ as
$$\mathcal{E}^{\mathsf{sp}}(X, Y) \cong \{g : K\,X \to K\,Y \mid X = g^*Y\} \qquad (1)$$

Using (1), we can reformulate the decoding of a δ code:

$$\begin{aligned}[\![\delta_A F]\!]\ Q &= \Sigma g : A \to K\,Q.\, [\![F(g^*Q)]\!]\ Q \\ &\cong \Sigma X : |\mathcal{E}_A|.\, \Sigma g : K\,X \to K\,Q.[\![F(X)]\!]Q \times (X = g^*Q) \\ &\cong \Sigma X : |\mathcal{E}_A|.\, \mathcal{E}^{\mathsf{sp}}(X, Q) \times [\![F(X)]\!]Q\end{aligned}$$

Here we use a constraint in the style of Henry Ford: choose 'any X you like as long as it is g^*Q'. It is now easy to see the action of $[\![\delta_A F]\!]$ on a splitting morphism $h : Q \to Q'$ after applying the isomorphism: It is a sum of pairs of actions, where we can go from $\mathcal{E}^{\mathsf{sp}}(X, Q)$ to $\mathcal{E}^{\mathsf{sp}}(X, Q')$ by composing with h, and from $[\![F(X)]\!]Q$ to $[\![F(X)]\!]Q'$ by the induction hypothesis. If h was an arbitrary morphism, there would be no guarantee that the composite would still be splitting.

The data type described by the fibred IR code c is the initial algebra of $[\![c]\!]$. We will show in Section III-D that this exists, under certain conditions on the fibration $K : \mathcal{E} \to \mathcal{B}$. But first, let us look at some examples of fibred inductive-recursive definitions.

C. Examples of Fibrational Induction-Recursion

We have shown that IR-codes c in $\mathsf{IR}\ K$ represent functors $[\![c]\!] : \mathcal{E}^{\mathsf{sp}} \to \mathcal{E}^{\mathsf{sp}}$ and hence data types using initial algebra semantics. Here are some examples. Of course all fibrations involved satisfy the conditions of Lemma 4.

Example 6. The families Fibration: We start with the families fibration $\mathsf{Fam}(|D|) \to \mathsf{Sets}$ from Example 3 because, when we specialise fibred induction-recursion to this fibration, we get a system of codes with exactly the same expressive power as Dybjer and Setzer's induction-recursion, i.e. they define exactly the same class of functors.

Firstly, let us look at the IR-codes themselves. There is a slight difference in the ι-codes as Dybjer and Setzer's ι-codes produce families with exactly one index. Our fibred IR-codes a priori are more general as they produce families with arbitrary index. However a family $T : U \to \mathsf{Sets}$ is actually the coproduct of the U-indexed set of families whose u'th family has index $\mathbf{1}$ which decodes to Tu, that is $(U, T) \cong \Sigma u : U.\ \ (\mathbf{1}, \lambda\_.Tu)$. Hence our ι-codes can be simulated by a Dybjer-Setzer σ-code built on top of ι-codes. Our σ-codes are exactly the same as Dybjer and Setzer's as are our δ-codes when we realise that a function $F : |\mathcal{E}_A| \to \mathsf{IR}\ K$ is exactly, in the families fibration, a function $F : (A \to D) \to \mathsf{IR}\ K$.

Next, when we look at the semantics of IR-codes, its clear that both sets of codes interpret ι as constant functors and σ as coproducts. As for the δ-constructor, these are seen to define the same functors once we realise that, in the families fibration, if we reindex a family $T : U \to \mathsf{Sets}$ by a function $g : U' \to U$, we have $g^*(U, T) = (U', T \circ g)$.

Before leaving the families fibration, we would like to make a final remark about the fibred presentation of induction-recursion. The presentation of the fibred IR-codes and their semantics seems conceptually cleaner and less notationally cumbersome – for example, the use of coproducts in the total category to give a uniform definition of $[\![-]\!]$, instead of a two phase definition using $[\![-]\!]^0$ and $[\![-]\!]^1$, highlights mathematical structure one might have missed. While the fibrational overhead is a price to pay, we think there is merit in an algebraic presentation of induction recursion to complement Dybjer and Setzer's type theoretic one. For most people however, we expect the merit of fibrational induction recursion to be its power to define new inductive-recursive structures by simply choosing new fibrations. We see an example of this now when constructing a universe of setoids.

Example 7. A Universe of Setoids: A setoid $(|A|, \simeq)$ in type theory is a type $|A|$ together with an equivalence relation $\simeq$ on $|A|$ (that is, a relation $\simeq$ together with proofs of reflexivity, transitivity and symmetry). Setoids are often used when developing mathematics in type theory [6], as they can simulate both quotient types and function extensionality. Naturally, we would like to consider universes also in this setting. By instantiating fibred induction-recursion in a fibration of families of setoids, we can get such universes without any more effort than if we were working with mere sets.

Following Palmgren [22], a family of setoids $B : A \to \mathsf{Setoid}$ consists of an index setoid $A = (|A|, \simeq)$ together with an $|A|$-indexed family of setoids $B_a : \mathsf{Setoid}$ for $a : |A|$ such that if p is a proof that $x \simeq y$, then B_x and B_y are "the same", i.e. there is a "reindexing" bijection $\phi_p : B_x \to B_y$ (in a coherent way). Setoids naturally form a category Setoid, with a morphism $f : (|A|, \simeq_A) \to (|B|, \simeq_B)$ being a function $|f| : |A| \to |B|$ which respects the equivalence relations. Families of setoids form a category $\mathsf{Fam}_{\mathsf{Setoid}}$ which is fibred over Setoid in the same way $\mathsf{Fam}(\mathsf{Sets})$ is fibred over Sets. In particular, the objects in the fibre $\mathcal{E}_A$ are families of setoids $B : A \to \mathsf{Setoid}$ with index setoid A.

Using fibred induction-recursion instantiated to this fibration, we can now write down a code for a universe of setoids closed under Σ-setoids:

$$c_{\mathbb{N},\Sigma} = \iota(\mathbf{1}, \lambda\_.\mathbb{N}) +_{\mathsf{IR}} \delta_{\mathbf{1}}(\lambda A.\delta_{A\star}(\lambda B.\iota(\mathbf{1}, \lambda\_.\Sigma_{\mathsf{Setoid}}\ (A\star)\ B)))$$

where $c +_{\mathsf{IR}} d \coloneqq \sigma_{\mathbf{2}}(\lambda x.\mathsf{if}\ x\ \mathsf{then}\ c\ \mathsf{else}\ d)$ encodes a binary coproduct of IR-codes. Here $\mathbf{1}$ and $\mathbb{N}$ are the unit setoid and setoid of natural numbers with the obvious equivalence relations respectively. For $A : \mathsf{Setoid}$ and $B : A \to \mathsf{Setoid}$, the sigma setoid is defined by $\Sigma_{\mathsf{Setoid}} A\ B = (\Sigma\, |A|\, |B|, \simeq_\Sigma)$ where $(x, y) \simeq_\Sigma (x', y')$ if $(\exists p : x \simeq_A x')(\phi_p(y) \simeq_{B_{x'}} y')$.

The code $c_{\mathbb{N},\Sigma}$ describes a universe (U, T) which contains the natural numbers and is closed under sigma setoids; the underlying family of sets $|U|, |T|$ satisfies the same equations as the universe of sets in the introduction, but all operations now automatically preserve the equivalence relations. Note that the code $c_{\mathbb{N},\Sigma}$ was obtained from the corresponding code for sets simply by replacing sets with their setoid equivalents in an entirely predictive way.

Example 8. The Relations Fibration: Within a relational version of induction-recursion, we are interested in defining a set-valued relation on a set X, that is a function $R : X \times X \to \mathsf{Sets}$. If from the existence of an element $p : R(x_1, x_2)$, we can construct new elements of X, then X cannot be defined before R and we are really dealing with an induction-recursion like structure rather than an instance of simpler indexed data types.

One example of a such a structure is Conway's ordered field of surreal numbers [10]. This is a field which contains the real numbers as a subfield and the ordinals as an ordered substructure, and as we will see, it is defined in a relational inductive-recursive way. Conway gave the following definition:

- A surreal number $X = (X_{\mathrm{L}}, X_{\mathrm{R}})$ consists of two sets X_{L} and X_{R} of surreal numbers, such that every element from X_{L} is smaller than any element from X_{R}. All surreal numbers are constructed this way.
- A surreal number $X = (X_{\mathrm{L}}, X_{\mathrm{R}})$ is greater than another surreal number $Y = (Y_{\mathrm{L}}, Y_{\mathrm{R}})$, $Y \leq X$, if and only if
  - there is no $x \in X_{\mathrm{R}}$ such that $x \leq Y$, and
  - there is no $y \in Y_{\mathrm{L}}$ such that $X \leq y$.

  This is an induction-recursion like structure, as the definition of the surreal numbers and the order relation on them needs to be simultaneous. In Agda, using families of types (with index sets from a universe U closed under the standard type formers, to keep the definition small) to model subsets, and bounded quantification $\forall[x \in X]\psi(x)$ for such subsets, we can give the following definition:

```
mutual
  data Surreal : Set where
    [[_|_]]_ :   (XL : Fam U Surreal) →
                 (XR : Fam U Surreal) →
                   ∀[ xl ∈ XL ](∀[ xr ∈ XR ] (xl ≤ xr))
                                → Surreal
  _≤_ : Surreal → Surreal → Set
```

$$
\begin{aligned}
&[\![\ YL \mid YR\ ]\!]\ p \leq [\![\ XL \mid XR\ ]\!]\ q \\
&\quad = \quad (\forall [\ x \in \quad XR\ ]\ (\neg\ (x \leq [\![\ YL \mid YR\ ]\!]\ p))) \\
&\quad \wedge \quad (\forall [\ y \in \quad YL\ ]\ (\neg\ ([\![\ XL \mid XR\ ]\!]\ q \leq y)))
\end{aligned}
$$

Example 9. The Category with Families Fibration: Categories with Families [17] were introduced by Dybjer as a syntax free representation of type theories. A category with families consists of a category $\mathcal{C}$ and a functor $F : \mathcal{C}^{\mathsf{op}} \to \mathsf{Fam}(\mathsf{Sets})$ with some extra structure. We think of $\mathcal{C}$ as a category of contexts, and of $F(\Gamma)$ as a family of terms, indexed by types, all in the context Γ. The (large) category of categories with families form a fibration with base category Cat, the category of all small categories and the fibre above $\mathcal{C}$ consists of the functors $F : \mathcal{C}^{\mathsf{op}} \to \mathsf{Fam}(\mathsf{Sets})$ with their extra structure.

An interesting category with families for a given type theory is the one formed from the syntax of the theory itself (the "term model"). The construction thereof for dependent type theories is famously induction-recursion like in nature; types are indexed over contexts, but contexts can only be extended by well-formed types. The syntax of dependent type theory has been implemented this way in Agda. However it should be pointed out that such constructions [8], [11] often use an inductive instead of recursive definition of types, which leads to the study of inductive-inductive definitions [21].

There are many similar examples: by using fibred induction-recursion instantiated to the presheaves fibration, we can define universes that automatically come equipped with a notion of substitution. We could even consider inductive-recursively defined fibrations themselves, since the category of fibrations is again fibred over Cat.

The next example is more than just an example, but rather shows the conceptual simplicity of fibrational induction-recursion. After having studied containers, one needed to study indexed containers. Dybjer and Setzer found a similar phenomenon in that after studying induction-recursion they had to study indexed induction-recursion [15] which allows one to define a family of universes $T_i : U_i \to D_i$ where I is an indexing set. Thus, perhaps we need to develop an "indexed" form of fibrational IR. This would somehow go against the grain as fibrations are a theory of indexing and so there should not be a need to index our fibrational induction-recursion. Fortunately this is not the case and so we have reduced the need for studying indexed induction-recursion by absorbing it within the unifying framework of fibrational induction-recursion.

Example 10. Dybjer and Setzer's Indexed IR: Let I by a set of indices for types D_i. Then Dybjer and Setzer's indexed induction-recursion arises as an example of fibred induction-recursion where the base category consists of I-indexed sets, i.e. the base category is $[I, \mathsf{Sets}]$, and the total category has as objects the I-indexed product of $\mathsf{Fam}(|D_i|)$. Note that rather pleasingly this is just the I-indexed product of the individual fibrations $\mathsf{Fam}(|D_i|) \to \mathsf{Sets}$.

In the same vein, we observe that both simple inductive types and inductive families in the form of containers and indexed containers are part of the fibred inductive-recursive framework.

Example 11. Containers and Indexed Containers: Fibred IR codes for the identity fibration $\mathsf{Id} : \mathsf{Sets} \to \mathsf{Sets}$ are exactly the codes for inductive definitions. The class of functors definable by these two schemes are therefore the same and they thus coincide with the class of functors definable by containers [20]. Indeed every code can be reduced to a "container normal form" $\sigma_S(\lambda s.\, \delta_{P\ s}\, \iota\, \mathbf{1})$.

Also indexed containers coincide with a class of fibred IR codes, this time for the domain fibration $\mathsf{dom}_I : \mathsf{Sets}/I \to \mathsf{Sets}$ where I is the fixed index set. Note that just as indexed containers, and indeed induction recursion can be presented with different input and outputs, so fibered induction recursion generalizes smoothly to allow for codes $\mathsf{IR}\ K\ K'$ where K and K' are different fibrations for input and output.

All of these examples have a similar fibrational structure, namely we start with categories $\mathbb{I}$ and $\mathbb{D}$ and a functor $F : \mathbb{I} \to \mathsf{Cat}$ and then seek to define universes of the form (U, T) where U is an object of $\mathbb{I}$ and $T : FU \to \mathbb{D}$ is a functor. Such universes are clearly fibered over $\mathbb{I}$. We will work at this level of abstraction in the journal version of this paper.

D. Existence of inititial algebras

We have shown that every IR code c gives rise to a functor $[\![c]\!] : \mathcal{E}^{\mathsf{sp}} \to \mathcal{E}^{\mathsf{sp}}$. In this section, we show that these functors indeed have initial algebras, under some conditions on the fibration $K : \mathcal{E} \to \mathcal{B}$. We do this by adapting Dybjer and Setzer's proof [13] to the fibrational setting. For the rest of this section, we assume that $K : \mathcal{E} \to \mathcal{B}$ is a split fibration, and that $\mathcal{E}^{\mathsf{sp}}$ has set-indexed coproducts.

Important for both their and our proof is keeping track of the "size" of the index objects A appearing in the δ codes. Crucially, these index objects may depend on the input object Q we pass to the functor. To make this precise, we collect all the index objects in a class $\mathsf{Aux}(c, Q)$:

Definition 6. For an IR code c and object Q in $\mathcal{E}$, define the collection $\mathsf{Aux}(c, Q) \subseteq |\mathcal{B}|$ by induction over c:

$$
\mathsf{Aux}(\iota(P), Q) = \emptyset \qquad \mathsf{Aux}(\sigma_A f, Q) = \bigcup_{a \in A} \mathsf{Aux}(fa, Q)
$$

$$
\mathsf{Aux}(\delta_A F), Q) = \{A\} \cup \bigcup_{g :\ A \to KQ} \mathsf{Aux}(F(g^* Q), Q)
$$

We now observe that if for certain Q, all $A \in \mathsf{Aux}(c, Q)$ are "small" in a suitable sense then $[\![c]\!]$ is κ-continuous, i.e. preserves κ-filtered colimits, for some regular κ. Hence, by a standard argument – see e.g. Adámek et al. [**?**] – we can conclude that $[\![c]\!]$ has an initial algebra as long as $\mathcal{E}^{\mathsf{sp}}$ has κ-filtered colimits and an inital object.

The categorical notion of smallness we use is κ-presentability. Recall that A is κ-presentable if $\mathcal{B}(A, -) : \mathcal{B} \to \mathsf{Sets}$ preserves κ-filtered colimits (see e.g. Adámek and Rosicky [4]). When $\mathcal{B}$ is Sets, an object A is κ-presentable if and only if it has cardinality $|A| < \kappa$, which is basically the set theoretic definition of smallness used by Dybjer and Setzer. We also require that every set of non-isomorphic β-presentable objects

of $\mathcal{B}$ has cardinality at most 2^{β}. Note that this is always true when $\mathcal{B}$ is Sets.

Lemma 5. *Let $\bigvee_i Q_i$ be a κ-filtered colimit for a diagram $Q : J \to \mathcal{E}^{\mathsf{sp}}$ with all $A \in \mathsf{Aux}(c, Q_i)$ κ-presentable. If $K^{\mathsf{sp}} : \mathcal{E}^{\mathsf{sp}} \to \mathcal{B}$ preserves $\bigvee_i Q_i$, then so does $[\![c]\!] : \mathcal{E}^{\mathsf{sp}} \to \mathcal{E}^{\mathsf{sp}}$.* ■

In order to ensure that the first hypothesis of the lemma holds we need a meta theoretical assumption, namely the existence of a Mahlo cardinal. This is not surprising, e.g. Dyber and Setzer needed this assumption for their system which ours can be instantiated to. Indeed within induction recursion we can build inaccessible universes and hence our meta theory will certainly involve large cardinals.

Recall that a cardinal M is a Mahlo cardinal if and only if it is inaccessible and every normal (i.e. strictly monotone and continuous at limit ordinals) function $f : \mathsf{M} \to \mathsf{M}$ has an inaccessible fixed point. We also assume that all indexing sets in the coproducts used in the decoding have cardinality at most M. This can be achieved by considering sets from V_{M} only, the level M of the Von Neumann hierarchy which is a model of ZFC since M is inaccessible – this is what Dybjer and Setzer did. Working inside this model allows us to prove that:

Lemma 6. *Let $c : \mathsf{IR}\ K$ and (Q_α) be the initial sequence of the induced functor $[\![c]\!]$. Then there exists an inaccessible κ such that all $A \in \mathsf{Aux}(c, Q_\alpha)$ are κ-presentable for all $\alpha < \kappa$.* ■

By combining Lemma 6 and Lemma 7, we get:

Theorem 1. *Under the assumptions of Lemma 7, if $\mathcal{E}^{\mathsf{sp}}$ has κ-filtered colimits for κ as in Lemma 7, and K^{sp} preserves them, then the functor $[\![c]\!] : \mathcal{E}^{\mathsf{sp}} \to \mathcal{E}^{\mathsf{sp}}$ has an initial algebra.* ■

Remark 2. In this section, we have assumed that $\mathcal{E}^{\mathsf{sp}}$ has coproducts. As remarked in Section III-B, we could replace coproducts with a functor $S : \mathsf{Fam}(\mathcal{E}^{\mathsf{sp}}) \to \mathcal{E}^{\mathsf{sp}}$. For Theorem 1 to go through, we need S to i) preserve colimits, and have the property that if (A, B) is a family with $|A| < \lambda$ and all $B(x)$ are λ-presentable, then so is $S(A, B)$; and ii) if A is a set, then $S(A, -)$ preserves filtered colimits. Importantly, the relations fibration satisfies these criteria even though its associated discrete fibration does not have coproducts. In the journal version of this paper we will expand upon this.

IV. Conclusions and Future Work

It is sad that induction-recursion remains a tool which few people use and even fewer understand. We believe that part of the problem is that induction-recursion needs an algebraic presentation to complement the type theoretic one presented by Dybjer and Setzer. We have presented such an algebraic framework for induction-recursion and believe that i) the fact that indexed induction-recursion as well as induction-recursion can be found within fibrational induction-recursion shows that the fibration perspective adds greater clarity to our understanding of induction recursion; ii) structures such as relational universes show that the fibrational framework extends Dybjer and Setzer's induction-recursion to a host of other structures; and iii) the axiomatic nature of our semantics shows that induction-recursion can be deployed in settings where set theoretic models are deemed not to suffice. We have shown that Dybjer and Setzer's model construction can be extended to construct initial algebras in many of those models.

As for the future, it is clear there is much to be done. We have recently shown that the functors definable using small induction-recursion, i.e. induction-recursion with the families fibration $\mathsf{Fam}(D) \to \mathsf{Sets}$ where D is a small set, are exactly the indexed containers [20]. Is there a corresponding fibred result, perhaps using small fibrations and the interpretation of indexed containers in locally Cartesian closed categories? What roles do fibred functors and algebraic set theory [19] play? Are Dybjer and Setzer's IR codes closed under composition as one would expect? And finally, how do we turn induction-recursion into standard programming languages technology?

Acknowledgment

This research is partially supported by EPSRC grants EP/C0608917/1, EP/G033374/1.

References

[1] M. Abbott, "Categories of containers," Ph.D. dissertation, University of Leicester, 2003.
[2] M. Abbott, T. Altenkirch, and N. Ghani, "Containers: Constructing strictly positive types," *TCS*, vol. 342, no. 1, pp. 3 – 27, 2005.
[3] J. Adámek, S. Milius, and L. Moss, "Initial algebras and terminal coalgebras: a survey," June 29 2010, draft.
[4] J. Adámek and J. Rosicky, *Locally Presentable and Accessible Categories*. Prentice-Hall, 1994.
[5] T. Altenkirch and P. Morris, "Indexed containers," in *LICS*, 2009, pp. 277 –285.
[6] G. Barthe, V. Capretta, and O. Pons, "Setoids in type theory," *Journal of Functional Programming*, vol. 13, no. 2, pp. 261–293, 2003.
[7] R. Bird and O. de Moor, *Algebra of programming*. Prentice-Hall, 1997.
[8] J. Chapman, "Type theory should eat itself," *Electronic Notes in Theoretical Computer Science*, vol. 228, pp. 21–36, 2009.
[9] J. Chapman, P.-É. Dagand, C. McBride, and P. Morris, "The gentle art of levitation," in *ICFP*, vol. 45, no. 9. ACM, 2010, pp. 3–14.
[10] J. Conway, *On numbers and games*. AK Peters, 2001.
[11] N. A. Danielsson, "A formalisation of a dependently typed language as an inductive-recursive family," *LNCS*, vol. 4502, pp. 93–109, 2007.
[12] P. Dybjer, "Inductive families," *Formal aspects of computing*, vol. 6, no. 4, pp. 440–465, 1994.
[13] P. Dybjer and A. Setzer, "A finite axiomatization of inductive-recursive definitions," in *TLCA*. Springer Verlag, 1999, pp. 129–146.
[14] ——, "Induction–recursion and initial algebras," *Annals of Pure and Applied Logic*, vol. 124, no. 1-3, pp. 1–47, 2003.
[15] ——, "Indexed induction–recursion," *Journal of logic and algebraic programming*, vol. 66, no. 1, pp. 1–49, 2006.
[16] N. Gambino and M. Hyland, "Wellfounded trees and dependent polynomial functors," in *Types for Proofs and Programs*, 2004, pp. 210–225.
[17] M. Hofmann, "Syntax and semantics of dependent types," in *Semantics and Logics of Computation*, 1997, pp. 79 – 130.
[18] B. Jacobs, *Categorical Logic and Type Theory*, ser. Studies in Logic and the Foundations of Mathematics. North Holland, 1999, vol. 141.
[19] A. Joyal and I. Moerdijk, *Algebraic Set Theory*. Cambridge University Press, 1995.
[20] L. Malatesta, T. Altenkirch, N. Ghani, P. Hancock, and C. McBride, "Small induction-recursion," in *TLCA*, 2013.
[21] F. Nordvall Forsberg and A. Setzer, "A finite axiomatisation of inductive-inductive definitions," in *Logic, Construction, Computation*, 2012, pp. 259 – 287.
[22] E. Palmgren, "Proof-relevance of families of setoids and identity in type theory," *Archive for Mathematical Logic*, vol. 51, pp. 35 – 47, 2012.