Positive Inductive-Recursive Definitions

Neil Ghani[1], Lorenzo Malatesta[1], and Fredrik Nordvall Forsberg[2]

[1] University of Strathclyde, UK
[2] Swansea University, UK

Abstract. We introduce a new theory of data types which allows for the definition of data types as initial algebras of certain functors $\mathsf{Fam}\,\mathbb{C} \to \mathsf{Fam}\,\mathbb{C}$. This theory, which we call *positive inductive-recursive definitions*, is a generalisation of Dybjer and Setzer's theory of inductive-recursive definitions within which $\mathbb{C}$ had to be discrete – our work can therefore be seen as lifting this restriction. This is a substantial endeavour as we need to not only introduce a type of codes for such data types (as in Dybjer and Setzer's work), but also a type of morphisms between such codes (which was not needed in Dybjer and Setzer's development). We show how these codes are interpreted as functors on $\mathsf{Fam}\,\mathbb{C}$ and how these morphisms of codes are interpreted as natural transformations between such functors. We then give an application of positive inductive-recursive definitions to the theory of nested data types. Finally we justify the existence of positive inductive-recursive definitions by adapting Dybjer and Setzer's set-theoretic model to our setting.

1 Introduction

Inductive types are the bricks of a dependently typed programming language: they represent the building blocks on which any other type is built. The mortar the dependently typed programmer has at her disposal for computation with dependent types is recursion. Usually, a type A is defined inductively, and then terms or types can be defined recursively over the structure of A. The theory of inductive-recursive definitions [7,8] explores the simultaneous combination of these two basic ingredients, pushing the limits of the theoretical foundations of data types.

The key example of an inductive-recursive definition is Martin-Löf's universe à la Tarski [19]. A type U consisting of codes for small types is introduced, together with a decoding function T, which maps codes to the types they denote. The definition is both inductive and recursive; the type U is defined inductively, and the decoding function T is defined recursively on the way the elements of U are generated. The definition needs to be simultaneous, since the introduction rules for U refer to T. We illustrate this by means of a concrete example: say we want to define a data type representing a universe containing a name for the natural numbers, closed under Σ-types. Such a universe will be the smallest family of sets (U, T) satisfying the following equations

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

In this definition we see how ground types and the type constructor Σ are reflected in U. The left summand of the right hand side of the equation defining U is a code for natural numbers, while the right summand is a code reflecting Σ-types. Indeed the name of a Σ-type, $\Sigma\, A\, B$ for $A : \mathsf{Set}$, $B : A \to \mathsf{Set}$, in the universe (U, T) will consists of a name in U for the type A, i.e. an element $u\!:\!U$, and a function $f : Tu \to U$ representing the A-indexed family of sets B. The decoding function T maps elements of U according to the description above: the code for natural numbers decodes to the set of natural numbers $\mathbb{N}$ while an element (u, f) of the right summand decodes to the Σ-type it denotes. Other examples of inductive-recursive definitions have also appeared in the literature, such as e.g. Martin-Löf's computability predicates [18] or Aczel's Frege structures [3]. Lately the use of inductive-recursive definitions to encode invariants in ordinary data structures has also been considered [11].

Dybjer's [7] insight was that these examples are instances of a general notion, which Dybjer and Setzer [8] later found a finite axiomatisation of. Their theory of inductive-recursive definitions IR consists of: (i) a representation of types as initial algebras of functors; (ii) a grammar for defining such functors. Elements of the grammar are called IR codes, while functors associated to IR codes are called IR functors. The theory naturally covers simpler inductive types such as lists, trees, vectors, red-black trees etc. as well. Dybjer and Setzer [9] then gave an initial algebra semantics for IR codes by showing that IR functors are naturally defined on the category $\mathsf{Fam}\, D$ of families of elements of a (possibly large) type D and that these functors do indeed have initial algebras. More generally, abstracting on the families construction and the underlying families fibration $\pi : \mathsf{Fam}\, D \to \mathsf{Set}$, we have recently shown how to interpret IR functors in an arbitrary fibration endowed with the appropriate structure [15]. In this article, we will only consider the families fibration.

There is, however, a complication. When interpreting IR functors such as those building universes closed under dependent products, the mixture of covariance and contravariance intrinsic in the Π operator forces one to confine attention to functors $\mathsf{Fam}\,|\mathbb{C}| \to \mathsf{Fam}\,|\mathbb{C}|$ or, equivalently, to work with only those morphisms between families which are commuting triangles. As we have shown [15], more abstractly, this corresponds to working in the split cartesian fragment of the families fibration $\pi : \mathsf{Fam}\,\mathbb{C} \to \mathsf{Set}$, i.e. to only consider those morphisms in $\mathsf{Fam}\,\mathbb{C}$ which represent strict reindexing. In this paper we remove this constraint and hence explore a further generalization of IR, orthogonal to the one proposed in Ghani et al. [15]. We investigate the necessary changes of IR needed to provide a class of codes which can be interpreted as functors $\mathsf{Fam}\,\mathbb{C} \to \mathsf{Fam}\,\mathbb{C}$. This leads us to consider a new variation IR^+ of inductive-recursive definitions which we call *positive inductive-recursive definitions*. The most substantial aspect of this new theory is that in order to define these new codes, one needs also to define the morphisms between those codes. This is no handle-turning exercise!

We first recall Dybjer and Setzer's theory of inductive-recursive definitions (Section 2). To develop the theory we then (i) introduce a syntax and semantics consisting of IR^+ codes and their morphisms, and an explanation how these codes are interpreted as functors $\mathsf{Fam}\,\mathbb{C} \to \mathsf{Fam}\,\mathbb{C}$, where $\mathbb{C}$ is an arbitrary category (Section 3); (ii) use positive inductive-recursive definitions to shed new light on nested data types (Section 4); (iii) give a detailed comparison with the existing theory of IR (Section 5); (iv) adapt Dybjer and Setzer's model construction to our setting (Section 6).

The paper uses a mixture of categorical and type theoretic constructions. However, the reader should bear in mind that the foundations of this paper are type theoretic. In other words, all constructions should be understood to take place in extensional Martin-Löf type theory with one universe Set. This is entirely standard in the literature. The one exception is the use of a Mahlo cardinal required to prove that positive inductive recursive functors have initial algebras in Section 6. It should be emphasised that the Mahlo cardinal is only used to justify the soundness of the theory, and does not play any computational role. We refer the interested reader to Dybjer and Setzer [8] – they use a Mahlo cardinal for the same purpose – for the technical details. We also use fibrational terminology occasionally when we feel it adds insight, but those not familiar with fibrations can simply ignore such comments.

2 Induction Recursion

The original presentation of induction recursion given by Dybjer [7] was as a schema. Dybjer and Setzer [8] further developed the theory to internalize the concept of an inductive-recursive definition. They developed a finite axiomatization of the theory through the introduction of a special type of codes for inductive-recursive definitions. The following axiomatization which closely follows Dybjer and Setzer [8] presents the syntax of IR as an inductive definition.

Definition 1 **(IR codes).** *Let D : type. The type of $\mathsf{IR}(D)$ codes has the following constructors:*

$$\frac{d : D}{\iota\, d \;:\; \mathsf{IR}(D)}$$

$$\frac{A : \mathsf{Set} \quad f : A \to \mathsf{IR}(D)}{\sigma_A f \;:\; \mathsf{IR}(D)}$$

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

This is the syntax of induction recursion – it is quite remarkable in our opinion that this most powerful of theories of data types can be presented in such a simple fashion. These rules have been written in natural-deduction style and we may use the ambient type theory to define, for example, the function f in the code $\sigma_A f$. An example of an IR code is given in Example 5 – this code represents the universe containing the natural numbers and closed under Σ-types given in Equation (1). We now turn to the semantics of induction recursion: we interpret IR codes as functors, and to this end, we use the standard families construction Fam from category theory. We start recalling the definition of the category $\mathsf{Fam}\,\mathbb{C}$ of families of objects of a category $\mathbb{C}$.

Definition 2. *Given a category $\mathbb{C}$, the category $\mathsf{Fam}\,\mathbb{C}$ has objects pairs (X, P) where X is a set and $P : X \to \mathbb{C}$ is a functor which we can think of as an X-indexed family of objects of $\mathbb{C}$. A morphism from (X, P) to (Y, Q) is a pair (h, k) where $h : X \to Y$ is a function, and $k : P \xrightarrow{\cdot} Q \circ h$ is a natural transformation.*

Of course, the naturality condition in the definition of a morphism of families is vacuous as the domains of the functors in question are discrete.

Remark 3. For any category $\mathbb{C}$, the category $\mathsf{Fam}\,\mathbb{C}$ always has rich structure:

- $\mathsf{Fam}\,\mathbb{C}$ is fibred over Set (see e.g. Jacobs [16]). We omit here the definitions, but recall the standard splitting cleavage of the fibration $\pi : \mathsf{Fam}\,\mathbb{C} \to \mathsf{Set}$ which is relevant later: a morphism $(h, k) : (X, P) \to (Y, Q)$ is a split cartesian morphism if k is a family of identity morphisms, i.e. if $Q = P \circ h$.
- $\mathsf{Fam}\,\mathbb{C}$ is the free set indexed coproduct completion of $\mathbb{C}$; that is $\mathsf{Fam}\,\mathbb{C}$ has all set indexed coproducts and there is an embedding $\mathbb{C} \to \mathsf{Fam}\,\mathbb{C}$ universal among functors $F : \mathbb{C} \to \mathbb{D}$ where $\mathbb{D}$ is a category with set indexed coproducts. Given an A-indexed collection of objects $(X_a, P_a)_{a:A}$ in $\mathsf{Fam}\,\mathbb{C}$, its A-indexed coproduct is the family $(\sum_{a:A} X_a, [P_a]_{a:A})$.
- $\mathsf{Fam}\,\mathbb{C}$ is cocomplete if and only if $\mathbb{C}$ has all small connected colimits (Carboni and Johnstone [6, dual of Prop. 2.1]).
- Fam is a functor $\mathsf{CAT} \to \mathsf{CAT}$; given $F : \mathbb{C} \to \mathbb{D}$, we get a functor $\mathsf{Fam}(F) : \mathsf{Fam}\,\mathbb{C} \to \mathsf{Fam}\,\mathbb{D}$ by composition: $\mathsf{Fam}(F)(X, P) = (X, F \circ P)$. Here CAT is the category of large categories.

When $\mathbb{C}$ is a discrete category every morphism between families (X, P) and (Y, Q) consists only of functions $h : X \to Y$ such that $P\,x = Q\,(h\,x)$ for all x in X. From a fibrational perspective, this amounts to the restriction to the split cartesian fragment $\mathsf{Fam}\,|\mathbb{C}|$ of the fibration $\pi : \mathsf{Fam}\,\mathbb{C} \to \mathsf{Set}$, for $\mathbb{C}$ an arbitrary category. This observation is crucial for the interpretation of IR codes as functors. Indeed, given a type D, which we think of as the discrete (possibly large) set of its terms, we interpret IR codes as functors $\mathsf{Fam}\,D \to \mathsf{Fam}\,D$.

Theorem 4 **(IR functors).** *Let D : type. Every code $\gamma : \mathsf{IR}(D)$ induces a functor*

$$[\![\gamma]\!] : \mathsf{Fam}\,D \to \mathsf{Fam}\,D$$

Proof. We define $[\![\gamma]\!] : \mathsf{Fam}\,D \to \mathsf{Fam}\,D$ by induction on the structure of the code. We first give the action on objects:

$$\begin{aligned}
[\![\iota\, c]\!](X, P) &= (1, \lambda_{\_}\,.\,c)\\
[\![\sigma_A\, f]\!](X, P) &= \sum_{a\,:A}[\![f\, a]\!](X, P)\\
[\![\delta_A\, F]\!](X, P) &= \sum_{g\,:A \to X}[\![F\,(P \circ g)]\!](X, P)
\end{aligned}$$

We now give the action on morphisms. Let $(h, \mathsf{id}) : (X, P) \to (Y, Q)$ be a morphism in $\mathsf{Fam}\,D$, i.e. $h : X \to Y$ and $Q \circ h = P$.

$$\begin{aligned}
[\![\iota\, c]\!](h, \mathsf{id}) &= (\mathsf{id}_1, \mathsf{id})\\
[\![\sigma_A\, f]\!](h, \mathsf{id}) &= [\mathsf{in}_a \circ [\![f\, a]\!](h, \mathsf{id})]_{a\,:A}\\
[\![\delta_A\, F]\!](h, \mathsf{id}) &= [\mathsf{in}_{h \circ g} \circ [\![F(Q \circ h \circ g)]\!](h, \mathsf{id})]_{g\,:A \to X}
\end{aligned}$$

Here, the last line type checks since $Q \circ h = P$, hence $Q \circ h \circ g = P \circ g$ and we can apply the induction hypothesis. $\square$

Note how the interpretation of both σ and δ codes makes essential use of coproducts of families as defined in Remark 3. In particular, the interpretation of a code $\delta_B F$ uses as index set of the coproduct the function space $(B \to X)$, which is a set since both B and X are.

Ghani et al. [14] introduces morphisms between IR codes. This makes $\mathsf{IR}(D)$ into a category, and the decoding $[\![-]\!] : \mathsf{IR}(D) \to [\mathsf{Fam}\,D, \mathsf{Fam}\,D]$ can be shown to be a full and faithful functor. We will draw inspiration from this in Section 3 when we generalise the semantics to endofunctors on $\mathsf{Fam}\,\mathbb{C}$ for possibly non-discrete categories $\mathbb{C}$.

We call a data type *inductive-recursive* if it is the initial algebra of a functor induced from an IR code. Let us look at some examples.

Example 5 (A universe closed under dependent sums). In the introduction, we introduced a universe in Equation (1), containing the natural numbers and closed under Σ-types, and claimed that this universe can be defined via an inductive-recursive definition. Indeed, one can easily write down a code $\gamma_{\mathbb{N},\Sigma} : \mathsf{IR}(\mathsf{Set})$ for a functor that will have such a universe as its initial algebra:

$$\gamma_{\mathbb{N},\Sigma} := \iota\,\mathbb{N} +_{\mathsf{IR}} \delta_1(X \mapsto \delta_{X*}(Y \mapsto \iota\,\Sigma(X*)\,Y)) \;: \mathsf{IR}(\mathsf{Set})$$

Here we have used $\gamma +_{\mathsf{IR}} \gamma' := \sigma_2\,(0 \mapsto \gamma; 1 \mapsto \gamma')$ to encode a binary coproduct as a 2-indexed coproduct. Also, in the above, note that $X : 1 \to \mathsf{Set}$ and so $X*$ is simply the application of X to the canonical element of 1. If we decode $\gamma_{\mathbb{N},\Sigma}$, we get a functor which satisfies

$$\begin{aligned}
[\![\gamma_{\mathbb{N},\Sigma}]\!](U, T) &\cong (1, \lambda\_.\mathbb{N}) + (\Sigma u{:}U\,.\,T(u) \to U, \lambda(u, f).\Sigma\, x{:}T(u)\,.\,T(f(x)))\\
&= (1 + \Sigma u{:}U\,.\,T(u) \to U, \mathsf{inl}\_ \mapsto \mathbb{N}; \mathsf{inr}(u, f) \mapsto \Sigma\, x{:}T(u)\,.\,T(f(x)))
\end{aligned}$$

so that the initial algebra (U, T) of $[\![\gamma_{\mathbb{N},\Sigma}]\!]$, which satisfies $(U, T) \cong [\![\gamma_{\mathbb{N},\Sigma}]\!](U, T)$ by Lambek's Lemma, indeed satisfies Equation (1).

Example 6 (A universe closed under dependent function spaces). In the same way, we can easily write a down a code for a universe closed under Π-types:

$$\gamma_{\mathbb{N},\Pi} \coloneqq \iota\,\mathbb{N} +_{\mathsf{IR}} \delta_1(X \mapsto \delta_{X*}(Y \mapsto \iota\,\Pi(X*)\,Y)) \ : \mathsf{IR}(\mathsf{Set})$$

Even though this looks extremely similar to the code in the previous example, we will see in the next section that there is a big semantic difference between them.

3 Positive Inductive-Recursive Definitions

Theorem 4 tells us that IR codes can be interpreted as functors on families built over a discrete category. What happens if we try to interpret IR codes on the category $\mathsf{Fam}\,\mathbb{C}$, and not just on the subcategory $\mathsf{Fam}\,|\mathbb{C}|$? The problem is that if we allow for more general morphisms, we can not prove functoriality of the semantics of a δ code as it stands anymore: it is essential to have an actual equality on the second component of a morphism in $\mathsf{Fam}\,\mathbb{C}$ in order to have a sound semantics (see Example 10 below).

In this section we propose a new axiomatization which enables us to solve this problem. This new theory, which we dub *positive inductive-recursive definitions*, abbreviated IR^+, represents a generalization of IR which allows the interpretation of codes as functors defined on $\mathsf{Fam}\,\mathbb{C}$.

3.1 Syntax and Semantics of $\mathsf{IR}^+(\mathbb{C})$

The crucial insight which guides us when introducing the syntax of IR^+ is to deploy proper functors in the introduction rule of a δ code. This enables us to remove the restriction on morphisms within inductive recursive definitions; indeed, if we know that $F : (A \to \mathbb{C}) \to \mathsf{IR}^+(\mathbb{C})$ is a <u>functor</u>, and not just a <u>function</u>, we do not have to rely on the equality $P \circ g = Q \circ h \circ g$ between objects in $\mathbb{C}^A$, but we can use the second component of a morphism (h, k) in $\mathsf{Fam}\,\mathbb{C}$ to get a map $P \circ g \to Q \circ h \circ g$; then we can use the fact that F is a functor to get a morphism between codes $F(P \circ g) \to F(Q \circ h \circ g)$.

But, now we have to roll up our sleeves. For $F : (A \to \mathbb{C}) \to \mathsf{IR}^+(\mathbb{C})$ to be a functor, we need both $A \to \mathbb{C}$ and $\mathsf{IR}^+(\mathbb{C})$ to be categories. While it is clear how to make $A \to \mathbb{C}$ a category, turning $\mathsf{IR}^+(\mathbb{C})$ into a category entails defining both codes and morphisms between codes simultaneously, in an inductive-inductive fashion [21]. We give an axiomatic presentation of IR^+ analogously to the one given in Section 2 for the syntax of IR; however we now have mutual introduction rules to build both the type of $\mathsf{IR}^+(\mathbb{C})$ codes and the type of $\mathsf{IR}^+(\mathbb{C})$ morphisms, for $\mathbb{C}$ a given category. The semantics we give then explains how $\mathsf{IR}^+(\mathbb{C})$ codes can be interpreted as functors on $\mathsf{Fam}\,\mathbb{C}$, while $\mathsf{IR}^+(\mathbb{C})$ morphisms between such codes can be interpreted as natural transformations.

Definition 7. *Given a category $\mathbb{C}$ we simultaneously define the type of positive inductive-recursive codes on $\mathbb{C}$, $\mathsf{IR}^+(\mathbb{C}) : \mathsf{type}$ and the type of morphisms between these codes $\mathsf{IR}^+(\mathbb{C})(\_,\_) : \mathsf{IR}^+(\mathbb{C}) \to \mathsf{IR}^+(\mathbb{C}) \to \mathsf{type}$ as follows:*

– $\mathsf{IR}^+(\mathbb{C})$ *codes:*

$$\frac{c : \mathbb{C}}{\iota\, c : \mathsf{IR}^+(\mathbb{C})}$$

$$\frac{A\ : \mathsf{Set} \qquad f\ : A \to \mathsf{IR}^+(\mathbb{C})}{\sigma_A f : \mathsf{IR}^+(\mathbb{C})}$$

$$\frac{A\ : \mathsf{Set} \qquad F : (A \to \mathbb{C}) \to \mathsf{IR}^+(\mathbb{C})}{\delta_A F : \mathsf{IR}^+(\mathbb{C})}$$

– $\mathsf{IR}^+(\mathbb{C})$ *morphisms:*

- *morphisms from ιc:*

$$\frac{f : \mathbb{C}(c, c')}{\mathit{\Gamma}_{\iota,\iota}(f) : \mathsf{IR}^+(\mathbb{C})(\iota\, c, \iota\, c')}$$

$$\frac{a : A \qquad r : \mathsf{IR}^+(\mathbb{C})(\iota\, c, fa)}{\mathit{\Gamma}_{\iota,\sigma}(a, r) : \mathsf{IR}^+(\mathbb{C})(\iota\, c, \sigma_A f)}$$

$$\frac{g : A \to \emptyset \qquad r : \mathsf{IR}^+(\mathbb{C})(\iota\, c, F(! \circ g))}{\mathit{\Gamma}_{\iota,\delta}(g, r) : \mathsf{IR}^+(\mathbb{C})(\iota\, c, \delta_A F)}$$

- *morphisms from $\sigma_A f$:*

$$\frac{\gamma, : \mathsf{IR}^+(\mathbb{C}) \qquad r : (a : A) \to \mathsf{IR}^+(\mathbb{C})(fa, \gamma)}{\mathit{\Gamma}_{\sigma,\gamma}(r) : \mathsf{IR}^+(\mathbb{C})(\sigma_A f, \gamma)}$$

- *morphisms from $\delta_A F$*

$$\frac{\rho : \mathsf{Nat}(F, \kappa_{\iota c})}{\mathit{\Gamma}_{\delta,\iota}(\rho) : \mathsf{IR}^+(\mathbb{C})(\delta_A F, \iota\, c)}$$

$$\frac{b : B \qquad \rho : \mathsf{Nat}(F, \kappa_{f\, b})}{\mathit{\Gamma}_{\delta,\sigma}(b, \rho) : \mathsf{IR}^+(\mathbb{C})(\delta_A F, \sigma_A f)}$$

$$\frac{g : B \to A \qquad \rho : \mathsf{Nat}(F, G(- \circ g))}{\mathit{\Gamma}_{\delta,\delta}(g, \rho) : \mathsf{IR}^+(\mathbb{C})(\delta_A F, \delta_B G)}$$

In the last three clauses we have indicated with $\kappa_\gamma : \mathbb{C}^A \to \mathsf{IR}^+(\mathbb{C})$ the constant functor with value γ.

We now explain how each code $\gamma : \mathsf{IR}^+(\mathbb{C})$ is interpreted as an endofunctor

$$[\![\gamma]\!] : \mathsf{Fam}\,\mathbb{C} \to \mathsf{Fam}\,\mathbb{C}$$

A functor which is isomorphic to a functor induced by an IR^+ code is called an IR^+ functor. The semantics of IR^+ closely follows the one given in Section 2; as before we make essential use of coproducts in $\mathsf{Fam}\,\mathbb{C}$. Having said that, the crucial feature which separates the semantics of IR^+ from the semantics of IR is the following: when explaining the semantics of IR we can first interpret IR codes as functors and only later we define morphisms between codes which are interpreted as natural transformations between the corresponding functors. In IR^+ the type of of codes and the type of morphisms between codes are simultaneously defined in an inductive-inductive way, and therefore they are also decoded simultaneously as functors and natural transformations respectively. This is exactly what the elimination principle for an inductive-inductive definition gives.

In the following theorem, note that there is no restriction on the category $\mathbb{C}$ – all structure that we need comes for free from the families construction Fam.

Theorem 8 **(IR^+ functors).** *Let $\mathbb{C}$ be an arbitrary category.*

(i) Every code $\gamma : \mathsf{IR}^+(\mathbb{C})$ induces a functor $[\![\gamma]\!] : \mathsf{Fam}\,\mathbb{C} \to \mathsf{Fam}\,\mathbb{C}$.
(ii) Every morphism $r : \mathsf{IR}^+(\mathbb{C})(\gamma, \gamma')$ for codes $\gamma, \gamma' : \mathsf{IR}^+(\mathbb{C})$ gives rise to a natural transformation $[\![r]\!] : [\![\gamma]\!] \stackrel{\cdot}{\longrightarrow} [\![\gamma']\!]$.

Proof. While the action on objects is the same for both IR^+ and IR functors, the action on morphisms is different when interpreting a code of type $\delta_A F$: in the semantics of IR^+ we exploit the fact that $F : (A \to \mathbb{C}) \to \mathsf{IR}^+(\mathbb{C})$ is now a functor by using its action on morphism (which we, for the sake of clarity, indicate with $F_\to$). We give the action of IR^+ functors on morphisms only, and refer to the semantics given in Theorem 4 for the action on objects of $\mathsf{Fam}\,\mathbb{C}$.

The action on morphisms is given as follows. Let $(h, k) : (X, P) \to (Y, Q)$ in $\mathsf{Fam}\,\mathbb{C}$. We define $[\![\gamma]\!](h, k) : [\![\gamma]\!](X, P) \to [\![\gamma]\!](Y, Q)$ by recursion on γ:

$$\begin{aligned}
[\![\iota\, c]\!](h, k) &= (\mathsf{id}_1, \mathsf{id}_c) \\
[\![\sigma_A f]\!](h, k) &= [\mathsf{in}_a \circ [\![f\, a]\!](h, k)]_{a:A} \\
[\![\delta_A F]\!](h, k) &= [\mathsf{in}_{h \circ g} \circ [\![F(Q \circ h \circ g)]\!](h, k) \circ [\![F_\to(g^*(k))]\!]_{(X,P)}]_{g:A \to X}
\end{aligned}$$

In the last clause $g^*(k) : P \circ g \stackrel{\cdot}{\longrightarrow} Q \circ h \circ g$ is the natural transformation with component $g^*(k)_a = k_{g\,a} : P(g\,a) \to Q(k(g\,a))$; note that such a natural transformation is nothing but the vertical morphism above A obtained by reindexing (id_X, k) along g in the families fibration $\pi : \mathsf{Fam}\,\mathbb{C} \to \mathsf{Set}$.

We now explain how a IR^+ morphism $r : \gamma \to \gamma'$ is interpreted as natural transformation $[\![r]\!] : [\![\gamma]\!] \stackrel{\cdot}{\longrightarrow} [\![\gamma']\!]$ between IR^+ functors by specifying the component $[\![r]\!]_{(X,P)}$ at $(X, P) : \mathsf{Fam}\,\mathbb{C}$. Naturality of these transformations can be proved by a routine diagram chasing.

$$\begin{aligned}
[\![\Gamma_{\iota,\iota}(f)]\!]_{(X,P)} &= (\mathsf{id}_1, f) \\
[\![\Gamma_{\iota,\sigma}(a,r)]\!]_{(X,P)} &= \mathsf{in}_a \circ [\![r]\!]_{(X,P)} \\
[\![\Gamma_{\iota,\delta}(g,r)]\!]_{(X,P)} &= \mathsf{in}_{!_X \circ g} \circ [\![r]\!]_{(X,P)} \\
[\![\Gamma_{\sigma,\gamma}(r)]\!]_{(X,P)} &= [[\![r\,a]\!]_{(X,P)}]_{a:A} \\
[\![\Gamma_{\delta,\iota}(\rho)]\!]_{(X,P)} &= [[\![\rho_{P\circ g}]\!]_{(X,P)}]_{g:A\to X} \\
[\![\Gamma_{\delta,\sigma}(b,\rho)]\!]_{(X,P)} &= \mathsf{in}_b \circ [[\![\rho_{P\circ g}]\!]_{(X,P)}]_{g:A\to X} \\
[\![\Gamma_{\delta,\delta}(f,\rho)]\!]_{(X,P)} &= [\mathsf{in}_{g\circ f} \circ [\![\rho_{P\circ g}]\!]_{(X,P)}]_{g:A\to X}
\end{aligned}$$

□

Example 9 (A universe closed under dependent sums in $\mathsf{Fam}\,\mathsf{Set}^{\mathsf{op}}$*).* In Example 5, we defined an ordinary IR code $\gamma_{\mathbb{N},\Sigma} : \mathsf{IR}(\mathsf{Set})$ for a universe closed under sigma types. We can extend this code to an IR^+ code

$$\gamma_{\mathbb{N},\Sigma} = \iota\,\mathbb{N} +_{\mathsf{IR}} \delta_1(X \mapsto \delta_{X*}(Y \mapsto \iota\,\Sigma(X*)\,Y)) : \mathsf{IR}^+(\mathsf{Set}^{\mathsf{op}})$$

where now $G :\equiv Y \mapsto \iota\,\Sigma(X*)\,Y$ and $F :\equiv X \mapsto \delta_{X*}\,G$ needs to be functors. Given $f : Y \to Y'$ in $X \to \mathsf{Set}^{\mathsf{op}}$, i.e. $f_x : Y(x) \to Y'(x)$ in $\mathsf{Set}^{\mathsf{op}}$, we have $\Sigma x : (X*).f_x : \Sigma(X*)\,Y \to \iota\,\Sigma(X*)\,Y'$ in $\mathsf{Set}^{\mathsf{op}}$ so that we can define

$$G(f) : \iota\,\Sigma(X*)\,Y \to \iota\,\Sigma(X*)\,Y'$$

by $G(f) = \Gamma_{\iota,\iota}(\Sigma x : (X*).f_x)$.

We also need F to be a functor. Given $f : X \to X'$ in $1 \to \mathsf{Set}^{\mathsf{op}}$, we need to define $F(f) : \delta_{X*}\,G \to \delta_{X'*}\,G$. According to Definition 7, it is enough to give $f_* : X'* \to X*$ and $[\mathsf{in}_{f_* x}]_{x:X'*}$, a natural transformation from G to $G(- \circ f_*)$. Notice that working in $\mathsf{Set}^{\mathsf{op}}$ made sure that f_* was going in the right direction.

Example 10 (A universe closed under dependent function spaces in $\mathsf{Fam}\,\mathsf{Set}^{\cong}$*).* In Example 6, we saw how we could use induction-recursion to define a universe closed under Π-types in $\mathsf{Fam}\,|\mathsf{Set}|$, using the following code:

$$\gamma_{\mathbb{N},\Pi} = \iota\,\mathbb{N} +_{\mathsf{IR}} \delta_1(X \mapsto \delta_{X*}(Y \mapsto \iota\,\Pi(X*)\,Y)) \ : \mathsf{IR}(\mathsf{Set})$$

If we try to extend this to an IR^+ code in $\mathsf{Fam}\,\mathsf{Set}$ or $\mathsf{Fam}\,\mathsf{Set}^{\mathsf{op}}$, we run into problems. Basically, given a morphism $f : X' \to X$, we need to construct a morphism $\Pi\,X'\,(Y \circ f) \to \Pi\,X\,Y$, which of course is impossible if e.g. $X' = 0$, $X = 1$, and $Y* = 0$.

Hence the inherent contravariance in the Π-type means that $\gamma_{\mathbb{N},\Pi}$ does not extend to a $\mathsf{IR}^+(\mathsf{Set})$ or $\mathsf{IR}^+(\mathsf{Set}^{\mathsf{op}})$ code. However, if we move to the groupoid $\mathsf{Set}^{\cong}$, which is the subcategory of Set with only isomorphisms as morphisms, we do get an $\mathsf{IR}^+(\mathsf{Set}^{\cong})$ code describing the universe in question, which is still living in a category beyond the strict category $\mathsf{Fam}\,|\mathsf{Set}|$. It would be interesting to understand the relevance of positive induction recursion to homotopy type theory.

4 Application: A Concrete Representation of Nested Types

Nested data types [2] have been used to implement a number of advanced data types in languages which support higher-kinded types, such as the widely-used functional programming language Haskell. Among these data types are those with constraints, such as perfect trees [22]; types with variable binding, such as untyped λ-terms [12]; cyclic data structures [13]; and certain dependent types [20].

A canonical example of a nested data type is `Lam` : $\mathsf{Set} \to \mathsf{Set}$ defined in Haskell as follows:

```
data Lam a = Var a | App (Lam a) (Lam a) | Abs (Lam (Maybe a))
```

The type `Lam a` is the type of untyped λ-terms over variables of type `a` up to α-equivalence. Here, the constructor `Abs` models the bound variable in an abstraction of type `Lam a` by the `Nothing` constructor of type `Maybe a`, and any free variable `x` of type `a` in an abstraction of type `Lam a` by the term `Just x` of type `Maybe a`; The key observation about the type `Lam a` is that elements of the type `Lam (Maybe a)` are needed to build elements of `Lam a` so that, in effect, the entire family of types determined by `Lam` has to be constructed simultaneously. Thus, rather than defining a family of inductive types, the type constructor `Lam` defines an *type-indexed inductive family of types.* The kind of recursion captured by nested types is a special case of *non-uniform recursion* [5].

This section asks the question *Are nested data types representable as containers?* There would be benefits of a positive answer in that one could then apply container technology to nested data types, e.g. one could classify the natural transformations between them and operate on them using, for example, the derivative. While the latter has clear practical importance, note that the canonical recursion operator `fold` associated to inductive types is, when analysed for nested data types, a natural transformation.

We give a positive answer to the above question using IR^+. We sketch our overall development as follows:

- we define a grammar Nest for defining nested types and a decoding function $(\!|-|\!) : \mathsf{Nest} \to [\mathsf{Set}, \mathsf{Set}] \to [\mathsf{Set}, \mathsf{Set}]$.
- We show that $(\!|N|\!)$ restricts to an endofunctor $(\!|N|\!)_{\mathsf{Cont}}$ on the category Cont of containers.
- We use IR^+ to define $(\!|N|\!)_{\mathsf{Cont}}$. Hence by the results of this paper, $(\!|N|\!)_{\mathsf{Cont}}$ has an initial algebra $\mu(\!|N|\!)_{\mathsf{Cont}}$. We finish by arguing that $\mu(\!|N|\!) = [\![\mu(\!|N|\!)_{\mathsf{Cont}}]\!]_{\mathsf{Cont}}$ and hence that, indeed, nested types are containers.

A Grammar for Nested Types. We now present a grammar for defining nested data types. It is not the most sophisticated grammar, since our point is not to push the theory of nested data types, but rather to illustrate an application of positive induction-recursion to nested data types. The grammar we use is

$$\mathcal{F} = \mathsf{Id} \mid K\,C \mid \mathcal{F} + \mathcal{F} \mid \mathcal{F} \times \mathcal{F} \mid \mathcal{F} \circledast \mathcal{F}$$

where C is any container. The intention is that Id stands for the identity functor mapping a functor to itself, KC stands for the constant functor mapping any functor to the interpretation of the container C, $+$ stands for the coproduct of functors, $\times$ for the product of functors and $\circledast$ for the pointwise composition of functors. These intentions are formalised by a semantics for the elements of our grammar given as follows

$$\begin{array}{ll}
(\!|-|\!) & : \ \mathsf{Nest} \to [\mathsf{Set}, \mathsf{Set}] \to [\mathsf{Set}, \mathsf{Set}] \\
(\!|\mathsf{Id}|\!)\ F & = F \\
(\!|K\ C|\!)\ F & = [\![C]\!]_{\mathsf{Cont}} \\
(\!|\mathcal{F} + \mathcal{F}_1|\!)\ F & = (\!|\mathcal{F}|\!)\ F + (\!|\mathcal{F}_1|\!)\ F \\
(\!|\mathcal{F} \times \mathcal{F}_1|\!)\ F & = (\!|\mathcal{F}|\!)\ F \times (\!|\mathcal{F}_1|\!)\ F \\
(\!|\mathcal{F} \circledast \mathcal{F}_1|\!)\ F & = (\!|\mathcal{F}|\!)\ F \circ (\!|\mathcal{F}_1|\!)\ F
\end{array}$$

For example, the functor

$$L\,F\,X \ = \ X \ + \ (FX \times FX) \ + \ F(X+1)$$

whose initial algebra is the type `Lam` is of the form $(\!|N_L|\!)$ where

$$N_L = Kid \ + \ (\mathsf{Id} \times \mathsf{Id}) \ + \ \mathsf{Id} \circledast (KM)$$

where id is the container with one shape and one position which represents the identity functor on Set, and M is the container with two positions having one shape above one position and no shapes above the other. M represents the functor on Set mapping X to $X + 1$.

Nested Types as Functors on Containers. The next thing on our agenda is to show that every element N of Nest has an interpretation as an operator on containers $(\!|N|\!)_{\mathsf{Cont}} : \mathsf{Cont} \to \mathsf{Cont}$.

$$\begin{tikzcd}
\mathsf{Cont} \arrow[r, "{[\![-]\!]_{\mathsf{Cont}}}"] \arrow[d, "{(\!|N|\!)_{\mathsf{Cont}}}"'] & {[\mathsf{Set}, \mathsf{Set}]} \arrow[d, "{(\!|N|\!)}"] \\
\mathsf{Cont} \arrow[r, "{[\![-]\!]_{\mathsf{Cont}}}"'] & {[\mathsf{Set}, \mathsf{Set}]}
\end{tikzcd}$$

This is done easily enough by recursion on N noting that containers are closed under coproduct, product and under composition. Thus, for example, if we define $(\!|N_L|\!)_{\mathsf{Cont}}(S, P)$ to be the container (S_L, P_L) then

$$\begin{array}{ll}
S_L & = 1 + (S \times S) + \Sigma s : S.\ Ps \to 2 \\
P_L\ (\mathsf{in}_1 *) & = 1 \\
P_L\ (\mathsf{in}_2\,(s, s')) & = Ps + Ps' \\
P_L\ (\mathsf{in}_3\,(s, f)) & = \Sigma p : Ps.\ \mathsf{if}\ fp\ \mathsf{then}\ 1\ \mathsf{else}\ 0
\end{array}$$

Nested Types are Containers. We know that $\mathsf{Cont} = \mathsf{Fam}\,\mathsf{Set}^{\mathsf{op}}$. Now, we want to show that for every code $N : \mathsf{Nest}$, the functor $(\!|N|\!)_{\mathsf{Cont}}$ is a IR^{+} functor: to see this one needs to carefully examine the constructions on families used to build $(\!|N|\!)_{\mathsf{Cont}}$. The only sophisticated construction is the use of Σ-types to model the composition operator used in the definition of nested types and seen in the definition of S_L and P_L. But, as we have seen in Example 9, families closed under Σ are canonical examples of a IR^{+} construction. Thus, by the results in Section 6, for every $N : \mathsf{Nest}$, the IR^{+} functor $(\!|N|\!)_{\mathsf{Cont}}$ has an initial algebra which is a container (S_N, P_N). Finally, since $[\![-]\!]_{\mathsf{Cont}}$ preserves initial objects and filtered colimits of cartesian morphisms ([1] Propositions 4.5.1 and 4.6.7) and we know from Lemma 14 in Section 6 that the initial algebra chain of an IR^{+} functor is made from cartesian morphisms only, we can conclude that $[\![(S_N, P_N)]\!]_{\mathsf{Cont}} = \mu(\!|N|\!)$ showing that all nested types indeed are definable using containers.

5 Comparison to Plain IR

We now investigate the relationship between IR^{+} and IR. Note that every type D can be regarded as a discrete category, which we by abuse of notation denote $|D|$. In the other direction, every category $\mathbb{C}$ gives rise to a type $|\mathbb{C}|$ whose elements are the objects of $\mathbb{C}$.

Proposition 11. *There is a function* $\varphi : \mathsf{IR}(D) \to \mathsf{IR}^{+}(|D|)$ *s.t.*

$$[\![\gamma]\!]_{\mathsf{IR}(D)} \cong [\![\varphi(\gamma)]\!]_{\mathsf{IR}^{+}(|D|)}$$

Proof. The only interesting case is the δ code. Since $|D|$ is a discrete category, also $A \to |D|$ is discrete. Hence a mapping on objects $(A \to |D|) \to \mathsf{IR}(D)$ can trivially be extended to a functor $(A \to |D|) \to \mathsf{IR}^{+}(|D|)$. □

This proposition shows that the theory of IR can be embedded in the theory of IR^{+}. In the next proposition we slightly sharpen this result. We use the functoriality of the Fam construction (Remark 3) to show that forgetting about the extra structure in IR^{+} simply gets us back to plain IR.

Proposition 12. *Let* $|-| : \mathsf{CAT} \to \mathsf{SET}$ *be the functor assigning to each category the collection of its objects. There is a function* $\psi : \mathsf{IR}^{+}\mathbb{C} \to \mathsf{IR}\,|\mathbb{C}|$ *such that*

$$\mathsf{Fam}\,|-| \circ [\![\gamma]\!]_{\mathsf{IR}^{+}\mathbb{C}} = [\![\psi(\gamma)]\!]_{\mathsf{IR}\,|\mathbb{C}|} \circ \mathsf{Fam}\,|-|$$

for all $\gamma : \mathsf{IR}^{+}\mathbb{C}$*. Furthermore,* $\psi \circ \varphi = \mathsf{id}$. □

6 Existence of Initial Algebras

We briefly revisit the initial algebra argument used by Dybjer and Setzer [8]. Inspecting their proof, we see that it indeed is possible to adapt it also for the more general setting of positive inductive-recursive definitions by making the appropriate adjustments.

Remember that we call a morphism $(h, k) : (U, T) \to (U, T')$ in $\mathsf{Fam}\,\mathbb{C}$ a *splitting morphism* if $k = \mathsf{id}_T$, i.e. $T' \circ h = T$. We indicate by $\mathsf{Fam}\,|\mathbb{C}|$ the subcategory (subfibration) of $\mathsf{Fam}\,\mathbb{C}$ with the same objects, but with morphisms the splitting ones only.

The proof of existence of initial algebras for IR functors as given by Dybjer and Setzer [8] takes place in the category $\mathsf{Fam}\,|\mathbb{C}|$. The hard work of the proof is split between two lemmas. First Dybjer and Setzer prove that an IR functor $[\![\gamma]\!]$ preserves κ-filtered colimits if κ is an inaccessible cardinal which suitably bounds the size of the index sets in the image of the filtered diagram. Secondly they use the assumption of the existence of a large cardinal, namely a Mahlo cardinal, to prove that such a cardinal bound for the index sets can actually be found. The exact definition of when a cardinal is a Mahlo cardinal will not be important for the current presentation; see Dybjer and Setzer [8] for how this assumption is used. The existence of an initial algebra then follows a standard argument: the initial algebra of a κ-continuous functor can be constructed as the colimit of the initial chain up to κ iterations (see e.g. Adámek et al. [4]).

Inspecting the proofs, we see that they crucially depend on morphisms being splitting in several places. Luckily, the morphisms involved in the corresponding proofs for IR^+ actually are! As is well-known, a weaker condition than κ-continuity is actually sufficient: it is enough that the functor in question preserve the specific colimit of the initial κ-chain. We thus show that the initial chain of a IR^+ functor actually lives in $\mathsf{Fam}\,|\mathbb{C}|$, which will allow us to modify Dybjer and Setzer's proof accordingly.

Lemma 13. *For every code $\gamma : \mathsf{IR}^+\,\mathbb{C}$ the induced functor $[\![\gamma]\!] : \mathsf{Fam}\,\mathbb{C} \to \mathsf{Fam}\,\mathbb{C}$ preserves splitting morphisms, i.e. if (f, g) is splitting, then so is $[\![\gamma]\!](f, g)$.*

Proof. By induction on the structure of the code. The interesting case is $\gamma = \delta_A F$. Let $(h, id) : (X, P \circ h) \to (Y, P)$ be a splitting morphism. We have

$$\begin{aligned} [\![\delta_A F]\!](h, \mathsf{id}) &= [in_{h\circ g} \circ [\![F(P \circ h \circ g)]\!](h, \mathsf{id}) \circ [\![F_{\to}(g^*(\mathsf{id}))]\!]_{(X,P)}]_{g\,:A\to X} \\ &= [in_{h\circ g} \circ [\![F(P \circ h \circ g)]\!](h, \mathsf{id})]_{g\,:A\to X} \end{aligned}$$

where $[\![F(g^*\mathsf{id})]\!]_{(X,P)} = \mathsf{id}$ since both g^*, F and $[\![\_]\!]$ are functors. By the induction hypothesis, each $[\![F(P \circ h \circ g)]\!](h, \mathsf{id})$ is splitting. Furthermore injections are splitting in $\mathsf{Fam}\,\mathbb{C}$. Since composition of splitting morphisms is still splitting and the cotuple of splitting morphisms is also splitting in $\mathsf{Fam}\,\mathbb{C}$ we conclude that $[\![\delta_A\, F]\!](h, \mathsf{id})$ is a splitting morphism. ☐

Lemma 14. *For each $\gamma : \mathsf{IR}^+\,\mathbb{C}$, the initial chain*

$$0 \to [\![\gamma]\!](0) \to [\![\gamma]\!]^2(0) \to \ldots$$

consists of splitting morphisms only.

Proof. Recall that the connecting morphisms $\omega_{j,k} : [\![\gamma]\!]^j(\mathsf{0}) \to [\![\gamma]\!]^k(\mathsf{0})$ are uniquely determined as follows:

– $\omega_{0,1} = !_{[\![\gamma]\!](\mathsf{0})}$ is unique.
– $\omega_{j+1,k+1}$ is $[\![\gamma]\!](\omega_{j,k}) : [\![\gamma]\!]([\![\gamma]\!]^j(\mathsf{0})) \to [\![\gamma]\!]([\![\gamma]\!]^k(\mathsf{0}))$.
– $\omega_{j,k}$ is the colimit cocone for j a limit ordinal.

We prove the statement by induction on j. It is certainly true that $!_{[\![\gamma]\!](\mathsf{0})} : (\mathsf{0}, !) \to [\![\gamma]\!](\mathsf{0})$ is an identity at each component – there are none. Thus $\omega_{0,1}$ is a splitting morphism. At successor stages, we can directly apply Lemma 13 and the induction hypothesis. Finally, at limit stages, we use the fact that the colimit lives in $\mathsf{Fam}\,|\mathbb{C}|$ and hence coincides with the colimit in that category on splitting morphisms, so that the colimit cocone is splitting. □

Inspecting Dybjer and Setzer's original proof, we see that it now goes through also for IR^+ if we insert appeals to Lemma 14 where necessary. To finish the proof, we also need to ensure that $\mathsf{Fam}\,\mathbb{C}$ has κ-filtered colimits; this is automatically true if $\mathbb{C}$ has all small connected colimits (compare Remark 3), since $\mathsf{Fam}\,\mathbb{C}$ then is cocomplete. Note that discrete categories have all small connected colimits for trivial reasons.

Theorem 15. *Assume that a Mahlo cardinal exists in the meta-theory. If $\mathbb{C}$ has connected colimits, then every functor $[\![\gamma]\!]$ for $\gamma : \mathsf{IR}^+\,\mathbb{C}$ has an initial algebra.* □

7 Conclusion

In this paper we have introduced the theory IR^+ of positive inductive-recursive definitions as a generalization of Dybjer and Setzer's theory IR of inductive-recursive definitions [8,9,10], different from the fibrational generalization explored in Ghani et al. [15]: by modifying both syntax and semantics of IR we have been able to broaden the semantics to all of $\mathsf{Fam}\,\mathbb{C}$ and not just $\mathsf{Fam}\,|\mathbb{C}|$. The theory of IR^+, with IR as a subtheory, paves the way to the analysis of more sophisticated data types which allow not only for the simultaneous definition of an inductive type X and of a recursive function $f : X \to D$, but also takes the intrinsic structure between objects in the target type D into account. This is the case for example when D is a setoid, the category Set or $\mathsf{Set}^{\mathsf{op}}$, a groupoid or, even more generally, an arbitrary category $\mathbb{C}$.

In future work we aim to explore the theory of IR^+ from a fibrational perspective: this will allow us to reconcile the theory of IR^+ with the analysis of IR as given in Ghani et al [15]. In particular this will amount to characterising the semantics of δ codes as left Kan extensions. Another interesting direction of research is to investigate to which extent the rich structure of the families construction Fam will help shed light on the analysis of IR^+ types: in particular to exploit the monadic structure of Fam and then to investigate the relationship between the theory of IR^+ and the theory of familial 2-functors introduced by Weber [17].

References

1. Abbott, M.: Category of Containers. Ph.D. thesis, University of Leicester (2003)
2. Abel, A., Matthes, R., Uustalu, T.: Iteration and coiteration schemes for higher-order and nested datatypes. Theoretical Computer Science 333(1-2), 3–66 (2005)
3. Aczel, P.: Frege structures and the notions of proposition, truth and set. In: Barwise, J., Keisler, H.J., Kunen, K. (eds.) The Kleene Symposium, Studies in Logic and the Foundations of Mathematics, vol. 101, pp. 31 – 59. Elsevier (1980)
4. Adámek, J., Milius, S., Moss, L.: Initial algebras and terminal coalgebras: a survey (June 29 2010), draft
5. Blampied, P.: Structured Recursion for Non-uniform Data-types. Ph.D. thesis, University of Nottingham (2000)
6. Carboni, A., Johnstone, P.: Connected limits, familial representability and Artin glueing. Mathematical Structures in Computer Science 5(04), 441–459 (1995)
7. Dybjer, P.: A general formulation of simultaneous inductive-recursive definitions in type theory. Journal of Symbolic Logic 65(2), 525–549 (2000)
8. Dybjer, P., Setzer, A.: A finite axiomatization of inductive-recursive definitions. In: Typed lambda calculi and applications: 4th international conference, TLCA'99, L'Aquila, Italy, April 7-9, 1999: proceedings. pp. 129–146. Springer Verlag (1999)
9. Dybjer, P., Setzer, A.: Induction–recursion and initial algebras. Annals of Pure and Applied Logic 124(1-3), 1–47 (2003)
10. Dybjer, P., Setzer, A.: Indexed induction–recursion. Journal of logic and algebraic programming 66(1), 1–49 (2006)
11. Ek, L., Holmström, O., Andjelkovic, S.: Formalizing Arne Andersson trees and Left-leaning Red-Black trees in Agda. Bachelor thesis, Chalmers University of Technology (2009)
12. Fiore, M., Plotkin, G., Turi, D.: Abstract syntax and variable binding. In: Proc. Logic in Computer Science. pp. 193–202 (1999)
13. Ghani, N., Hamana, M., Uustalu, T., Vene, V.: Representing cyclic structures as nested types (2006), presented at Trends in Functional Programming
14. Ghani, N., Hancock, P., Malatesta, L., McBride, C., Altenkirch, T.: Small induction recursion. In: TLCA 2013 (2013)
15. Ghani, N., Malatesta, L., Nordvall Forsberg, F., Setzer, A.: Fibred data types. In: LICS 2013 (2013)
16. Jacobs, B.: Categorical Logic and Type Theory, Studies in Logic and the Foundations of Mathematics, vol. 141. North Holland, Elsevier (1999)
17. Mark, W.: Familial 2-functors and parametric right adjoints. Theory and Applications of Category Theory 18(22), 665–732 (2007)
18. Martin-Löf, P.: An intuitionistic theory of types (1972), published in Twenty-Five Years of Constructive Type Theory
19. Martin-Löf, P.: Intuitionistic type theory. Bibliopolis Naples (1984)
20. McBride, C., McKinna, J.: The view from the left. Journal of Functional Programming 14(1), 69–111 (2004)
21. Nordvall Forsberg, F., Setzer, A.: A finite axiomatisation of inductive-inductive definitions. In: Berger, U., Hannes, D., Schuster, P., Seisenberger, M. (eds.) Logic, Construction, Computation, Ontos mathematical logic, vol. 3, pp. 259 – 287. Ontos Verlag (2012)
22. Ralf, H.: Functional pearl: Perfect trees and bit-reversal permutation. Journal of Functional Programming 10(3), 305–317 (2000)