Small Induction Recursion

Peter Hancock[2], Conor McBride[2], Neil Ghani[2], Lorenzo Malatesta[2], and Thorsten Altenkirch[1]

[1] University of Nottingham*
[2] University of Strathclyde**

Abstract. There are several different approaches to the theory of data types. At the simplest level, polynomials and containers give a theory of data types as free standing entities. At a second level of complexity, dependent polynomials and indexed containers handle more sophisticated data types in which the data have an associated indices which can be used to store important computational information. The crucial and salient feature of dependent polynomials and indexed containers is that the index types are defined in advance of the data. At the most sophisticated level, induction-recursion allows us to define data and indices simultaneously.

This work investigates the relationship between the theory of *small* inductive recursive definitions and the theory of dependent polynomials and indexed containers. Our central result is that the expressiveness of small inductive recursive definitions is exactly the same as that of dependent polynomials and indexed containers. A second contribution of this paper is the definition of morphisms of small inductive recursive definitions. This allows us to extend our main result to an equivalence between the category of small inductive recursive definitions and the category of dependent polynomials/indexed containers. We comment on both the theoretical and practical ramifications of this result.

1 Introduction

One of the most important concepts in computer science is the notion of an inductive definition. It is difficult to trace back its origin since this concept permeates the history of proof theory and a large part of theoretical computer science. In recent years, the desire to explore, understand, and extend the concept of an inductive definition has led different researchers to different but (extensionally) equivalent notions. The theory of containers [1], and polynomial functors [18,12] are some of the outcomes of this research These theories give a comprehensive account of those data types such as `Nat` (the natural numbers), `List a` (lists containing data of a given type `a`), and `Tree a` (trees containing, once more, data of a given type `a`) which are free-standing in that their definition does not require the definition of other inter-related data types.

* Supported by a grant from the Institute of Advanced Studies, Princeton, and EPSRC grant EP/G03298X/1.
** Supported by EPSRC grant EP/G033056/1.

M. Hasegawa (Ed.): TLCA 2013, LNCS 7941, pp. 156–172, 2013.

These theories are too simple to capture more sophisticated data types possessing features such as: (*i*) variable binding as in the untyped and typed lambda calculus; (*ii*) constraints as in red black trees; and (*iii*) extra information about data having such types - e.g. vectors which record the lengths of lists. Therefore containers and polynomials have been generalised to indexed containers [2] and dependent polynomials [12,13] to capture not only free standing data types such as those mentioned above, but also data types where the data are indexed by an index storing computationally relevant information. Containers and (non-dependent) polynomials arise as specific instances of these generalised notions where the type of indices is chosen to be a singleton type.

However, even dependent polynomials and indexed containers fail to cover all data types of interest as they require the indices to be defined before the data. Induction-recursion (IR), developed in the seminal works of Peter Dybjer and Anton Setzer [9,10,11], remedies this deficiency. The key feature of an inductive-recursive definition is the simultaneous inductive definition of a small type X of indices together with the recursive definition of a function $T : X \to D$ from X into a type D which may be large or small. Since X and T can be defined at the same time, the indices need not be defined in advance of the data. Universes (introduced by Martin-Löf in the early 70's [17]) are paradigm examples of inductive recursive definitions.

It is natural to ask what is the relationship between dependent polynomials and induction recursion. Can we characterise those inductive-recursive definitions which correspond to dependent polynomials? This paper makes the following concrete contributions: i) we show that dependent polynomial and indexed containers correspond exactly to small inductive-recursive definitions, where "smallness" refers to the size of the target-type D; ii) we define morphisms of small inductive recursive definitions and use this notion to show that the resulting category of small inductive-recursive definitions is equivalent to the category of dependent polynomials and indexed containers; and iii) we extend these results to cover small indexed induction recursion.

These results have theoretical and practical importance. At the theoretical level, while it has been conjectured that the power of induction recursion lies in the case where D is large, no formal proof exists before this paper. Further, we contribute to the theory of induction recursion by giving a notion of morphism between inductive recursive definitions in the same way that containers, indexed containers and dependent polynomials have morphisms. Finally, dependent polynomials and indexed containers have a rich algebraic structure so our work shows that structure can be transported to Small IR - see the conclusion for details. Note that this structure is defined by universal properties and hence its transportation would not be possible without the work in Section 5 on morphisms. At a practical level, while systems such as Agda accept induction recursion, some systems, eg Coq, do not. This work gives a simple way to add small induction recursion to Coq by showing how to translate such definitions into indexed containers. It also allows programmers to convert definitions between the two forms, according to which works better for their own applications. To achieve this, and

to make the paper more accessible and non-hermetic, and to type check our translations, we have implemented our translations in Agda and provide lots of Agda examples.

The paper is organised as follows: in Section 2 we set our notation, while Section 3 recalls indexed containers, dependent polynomials and induction recursion. In Section 4, we show an equivalence between data types definable by small IR and those data types definable using dependent polynomials and/or indexed containers. In Section 5 we introduce the category of small inductive-recursive definitions and show it equivalent to the category of dependent polynomials/indexed containers. In Section 6 we briefly recall the theory of indexed inductive-recursive definitions, and extend the previous equivalence to the case of indexed small induction recursion. We conclude in Section 7.

The sources, proofs and additional materials for this paper are available from `http://personal.cis.strath.ac.uk/~conor/pub/SmallIR`.

2 Preliminaries and Internal Languages

We follow the standard approach of using extensional Martin-Löf type theory as the internal language to formalise reasoning with the locally cartesian closed structure of the category of sets — see [19,14] for details [1]. Our notation follows Agda — indeed, this paper is a literate Agda development. We write identity types as $x \equiv y$ and assume uniqueness of identity proofs. We write ΣT or $(s : S) \times T\ s$ and ΠT or $(s : S) \to T\ s$ for the dependent sum and dependent product in Martin-Löf type theory of $T : S \to \mathsf{Set}$. The elements of $(s : S) \times T\ s$ are pairs (s, t) where $s : S$ and $t : T\ s$ may be projected by π_0 and π_1. The elements of $(s : S) \to T\ s$ are functions $\lambda\, x \to t\ x$ mapping each element $s : S$ to an element $t\ s$ of $T\ s$.

Categorically, we think of an I-indexed type as a morphism $f : X \to I$ with codomain I. These are objects of the slice category Set/I. Morphisms in Set/I from object $f : X \to I$ to object $f' : X' \to I$ are given by functions $h : X \to X'$ such that $f = f' \circ h$. Type theoretically, we can represent matters in more or less the same way – that is, an object in a slice Set/I is a pair (X, f) of a set X (the domain), and a $\mathsf{function}$ $f : X \to I$. However, another possibility is to model an I-indexed type by a function $F : I \to \mathsf{Set}$ where $F\ i$ represents the fibre of f above i, i.e. as $(X, f)^{-1}\ i$, defined as follows.

$$\begin{array}{ll} {\cdot}^{-1} : \mathsf{Set}/I \to (I \to \mathsf{Set}) & \exists. : (I \to \mathsf{Set}) \to \mathsf{Set}/I \\ (X, f)^{-1}\ i = (x : X) \times f\ x \equiv i & \exists.F = (\Sigma F, \pi_0) \end{array}$$

We write $\exists.F$ for the inverse of this operator: that these are inverse (given uniqueness of identity proofs) is at the heart of the well known equivalence between the categories Set/I and $I \to \mathsf{Set}$ which, in a sense, underlies the equivalences we describe in this paper.

[1] The correspondence between lcccs and Martin Löf type theories is affected by coherence problems related to the interpretation of substitution. We refer to [7], [14] and more recently [5] for different solutions to these problems.

Given a function $k : I \to J$, we can form three very important functors. The pullback along k of an object $f : X \to J$ of Set/J defines a *reindexing* functor $\Delta_k : \mathsf{Set}/J \to \mathsf{Set}/I$. Δ_k has both a left adjoint and a right adjoint, respectively Σ_k,$\Pi_k : \mathsf{Set}/I \to \mathsf{Set}/J$. In the internal language, we define these for $\cdot \to \mathsf{Set}$, as follows:

$$\begin{array}{ll} \Delta_k : (J \to \mathsf{Set}) \to (I \to \mathsf{Set}) & \Sigma_k : (I \to \mathsf{Set}) \to (J \to \mathsf{Set}) \\ \Delta_k\ F\ i = F\ (k\ i) & \Sigma_k\ F\ j = (i\!:\!I) \times k\ i \equiv j \times F\ i \\ & \Pi_k : (I \to \mathsf{Set}) \to (J \to \mathsf{Set}) \\ & \Pi_k\ F\ j = (i\!:\!I) \to k\ i \equiv j \to F\ i \end{array}$$

3 Three Theories of Data Types

The foundation of our understanding of data types is initial algebra semantics. Thus, formally our theories of data types are in fact theories of functors which have initial algebras. In this section we recall the notions of dependent polynomials, indexed containers and induction recursion, each of which define certain classes of functors and hence data types.

Definition 1. *The collection of* dependent polynomials *with input indices I and output indices O is written* $\mathsf{Poly}\ I\ O$ *and consists of triples (r, t, q) where $I \leftarrow^r P \to^t S \to^q O$. A* dependent polynomial functor *is any functor isomorphic to some $[\![(r, t, q)]\!]_{\mathsf{Poly}} = \Sigma_q \circ \Pi_t \circ \Delta_r : \mathsf{Set}/I \to \mathsf{Set}/O$, illustrated as follows:*

$$\mathsf{Set}/I \xrightarrow{\Delta_r} \mathsf{Set}/P \xrightarrow{\Pi_t} \mathsf{Set}/S \xrightarrow{\Sigma_q} \mathsf{Set}/O\ .$$

While the definition above is concise, some readers may prefer a more concrete presentation. So we turn to the representation of dependent polynomials in the internal language. This leads us to the notion of an *indexed container*.

Definition 2. Indexed containers *with input indices I and output indices O is written* $\mathsf{IC}\ I\ O$ *and consists of triples (S, P, n) where $S : O \to \mathsf{Set}$, $P : (o\!:\!O) \to S\ o \to \mathsf{Set}$ and $n : (o\!:\!O) \to (s\!:\!S\ o) \to P\ o\ s \to I$. Its extension is the functor*

$$\begin{array}{l} [\![\cdot]\!]_{\mathsf{IC}} : \mathsf{IC}\ I\ O \to (I \to \mathsf{Set}) \to (O \to \mathsf{Set}) \\ [\![(S, P, n)]\!]_{\mathsf{IC}}\ X\ o = (s\!:\!S\ o) \times (p\!:\!P\ o\ s) \to X\ (n\ o\ s\ p) \end{array}$$

Every dependent polynomial functor (r, t, q) gives rise to an indexed container $(\hat{S}, \hat{P}, n)$.

$$\begin{array}{ll} \hat{S}\ o & = (S, q)^{-1}\ o \\ \hat{P}\ o\ (s, \_) & = (P, t)^{-1}\ s \\ n\ o\ (s, \_)\ (p, \_) & = r\ p \end{array}$$

We may readily check that

$$\begin{array}{rl} [\![(\hat{S}, \hat{P}, n)]\!]_{\mathsf{IC}}\ F\ o = & (sq\!:\!((S, q)^{-1}\ o)) \times (pq\!:\!((P, t)^{-1}\ (\pi_0\ sq))) \to F\ (r\ (\pi_0\ pq)) \\ \cong & (s\!:\!S) \times (q\ s \equiv o) \times (p\!:\!P) \to (t\ s \equiv p) \to F\ (r\ p) \\ = & (\Sigma_q \circ \Pi_t \circ \Delta_r)\ F\ o \end{array}$$

confirming the equivalence between indexed containers and dependent polynomials.

Polynomials (resp. containers) arise as a special case of dependent polynomials (indexed containers) by choosing $I = O = 1$. Notice the salient feature of both dependent polynomials and indexed containers — that the input and output indices I and O are fixed and must be defined in advance. This restriction means that neither dependent polynomials nor indexed containers suffice to define all the data types in which we are interested. Paradigmatic undefinable data types are universes of types. These are pairs (U, T) consisting of a set U, thought as a set of names or codes, and of a function $T : U \to Set$, thought as a "decoding function" which assigns a set $T\,u$ to every element u of U. For example, consider a universe containing the type of natural numbers $\mathbb{N}$ and closed under Σ-types. Such a universe will be the least solution of the

$$\begin{array}{lcl} U & = & \mathsf{1} + (u\!:\!U) \times T\ u\ \to\ U \\ T\ (\mathsf{inl}\ \star) & = & \mathbb{N} \\ T\ (\mathsf{inr}\ (u, f)) & = & (x\!:\!T\ u) \times T\ (f\ x) \end{array}$$

Note how, in this example, the set of codes U must be defined simultaneously with the decoding function T - something not possible with dependent polynomials or indexed containers which require that U be defined before T. Dybjer and Setzer developed the theory of induction recursion to cover exactly such inductive definitions where the indices and the data must be defined simultaneously. The first presentation of induction-recursion [8] was as an external schema. In later presentations [9,10], inductive recursive definitions are given via a type of codes $\mathsf{IR}\ I\ O$. [2]

Definition 3. *Let I, O be types. The type of* $\mathsf{IR}\ I\ O$*-codes has the following constructors*

$$\begin{array}{l} \textbf{data}\ \mathsf{IR}\ (I\ O\ :\ \mathsf{Set})\ :\ \mathsf{Set1}\ \textbf{where} \\ \quad \iota\ :\ (o\ :\ O) \qquad\qquad\qquad\qquad\qquad\qquad \to\ \mathsf{IR}\ I\ O \\ \quad \sigma\ :\ (S\ :\ \mathsf{Set})\ (K\ :\ S \qquad \to\ \mathsf{IR}\ I\ O)\ \to\ \mathsf{IR}\ I\ O \\ \quad \delta\ :\ (P\ :\ \mathsf{Set})\ (K\ :\ (P\ \to\ I)\ \to\ \mathsf{IR}\ I\ O)\ \to\ \mathsf{IR}\ I\ O \end{array}$$

In general I and O may be large types such as Set *or* $\mathsf{Set} \to \mathsf{Set}$ *etc. Above, we encode* small *induction recursion (small* IR*) we mean the cases where I and O are sets.*

Dybjer and Setzer [9,10] prove that every IR code defines a functor. In the case of *small* IR, this functorial semantics can be given in terms of slice categories. Before giving this semantics, we note that slice categories have set-indexed coproducts. That is, given a set A, and an A-indexed collection of objects $f_a : X_a \to I$ of Set/I, the cotuple $[f_a]_{a:A} : \coprod_{a:A} X_a \to I$ is the coproduct of the objects f_a in Set/I. We use $\mathsf{in}_a : X_a \to \coprod_{a:A} X_a$ for the a-th injection. In the internal

[2] Dybjer and Setzer treated only the case where I and O are the same. Our mild generalization allows the construction of partial fixed points.

language, the coproduct of an A-indexed family $X_a : I \to \mathsf{Set}$ is the function mapping i to $(a\!:\!A) \times X_a\ i$. We use these coproducts to give a definition of the functor denoted by an IR code more compact than - but of course equivalent to - that originally provided by Dybjer and Setzer.

Definition 4. *Let I, O be sets, $\gamma : \mathsf{IR}\ I\ O$. The action of the functor $[\![\gamma]\!] : \mathsf{Set}/I \to \mathsf{Set}/O$ on an object $f : X \to I$ of Set/I is defined by recursion on γ as follows*

- *if $\gamma = \iota\ o$ for some $o : O$*

$$[\![\iota\ o]\!]\,(f : X \to I) = (\lambda\_.o) : 1 \to O$$

- *if $\gamma = \sigma\ S\ K$ for some $S : \mathsf{Set}$, $K : S \to \mathsf{IR}\ I\ O$*

$$[\![\sigma\ S\ K]\!]\,(f : X \to I) = \coprod_{s:S} [\![K\ s]\!]\,f$$

- *if $\gamma = \delta\ P\ K$ for some $P : \mathsf{Set}$, $K : (P \to I) \to \mathsf{IR}\ I\ O$*

$$[\![\delta\ P\ K]\!]\,(f : X \to I) = \coprod_{x:P\to X} [\![K(f \circ x)]\!]\,f$$

An IR functor is any functor isomorphic to one of the form $[\![\gamma]\!]$ for some $\gamma : \mathsf{IR}IO$.

We can give the above construction in type theory, using the *direct* translation of slices, closed under dependent sum, yielding an interpretation in the style of Dybjer and Setzer:

$$\begin{array}{l}
[\![\cdot]\!]_{\mathsf{DS}} : \mathsf{IR}\ I\ O \to \mathsf{Set}/I \to \mathsf{Set}/O \\
[\![\iota\ o]\!]_{\mathsf{DS}}\ \ (X,f) = (1, \lambda\ _ \to o) \\
[\![\sigma\ S\ K]\!]_{\mathsf{DS}}\ (X,f) = (s\!:\!S) \times [\![K\ s]\!]_{\mathsf{DS}}\ (X,f) \\
[\![\delta\ P\ K]\!]_{\mathsf{DS}}\ (X,f) = (x\!:\!P \to X) \times [\![K\ (f \circ x)]\!]_{\mathsf{DS}}\ (X,f)
\end{array}$$

For any $\gamma : \mathsf{IR}\ I\ I$, we can construct of an inductive datatype simultaneously with its recursive decoder as the initial algebra, $((\mu\ \gamma, \mathsf{decode}\ \gamma), \mathsf{in})$, of $[\![\gamma]\!]_{\mathsf{DS}}$.

$$\begin{array}{l}
\textbf{data}\ \mu\ (\gamma : \mathsf{IR}\ I\ I) : \mathsf{Set}\ \textbf{where} \\
\quad \mathsf{in} : \mathsf{dom}\ ([\![\gamma]\!]_{\mathsf{DS}}\ (\mu\ \gamma, \mathsf{decode}\ \gamma)) \to \mu\ \gamma \\
\\
\mathsf{decode} : (\gamma : \mathsf{IR}\ I\ I) \to \mu\ \gamma \to I \\
\mathsf{decode}\ \gamma\ (\mathsf{in}\ t) = \mathsf{fun}\ ([\![\gamma]\!]_{\mathsf{DS}}\ (\mu\ \gamma, \mathsf{decode}\ \gamma))\ t
\end{array}$$

Here dom computes the domain of a universe and fun the decoder of a universe. As an example, we show that all containers [1] can be defined by induction recursion:

Example 1 (containers and W-types). Given a simple container (S, P), where $S : \mathsf{Set}$ and $P : S \to \mathsf{Set}$, we can represent it by an $\mathsf{IR}\ 1\ 1$ code as follows:

$$\begin{array}{l}
\mathsf{cont} : (S : \mathsf{Set}) \to (P : S \to \mathsf{Set}) \to \mathsf{IR}\ 1\ 1 \\
\mathsf{cont}\ S\ P = (\sigma\ S\ \lambda\ s \to \delta\ (P\ s)\ \lambda\ p \to \iota\ \star)
\end{array}$$

We note that $\mathsf{dom}\ [\![\mathsf{cont}\ S\ P]\!]_{\mathsf{DS}}\ (X, \_) = (s : S) \times (P\ s \to X) \times 1$ and that $\mu\ (\mathsf{cont}\ S\ P)$ thus amounts to Martin-Löf's well-ordering type $\mathsf{W}\ S\ P$. As a corollary of our main result we shall see that $\mathsf{IR}\ 1\ 1$ codes describe exactly the category of containers and their morphisms.

Example 2 (A Language of Sums and Products). If $\mathsf{Fin} : \mathbb{N} \to \mathsf{Set}$ maps n to a set with n elements, we can implement finitary summation and product with types:

$$\mathsf{sum}\ \mathsf{prod} : (n{:}\mathbb{N}) \to (\mathsf{Fin}\ n \to \mathbb{N}) \to \mathbb{N}$$

Next we can encode a datatype of numerical expressions closed under constants, sums and products, where each expression decodes to its numerical value — we need to know these values to compute the correct domains for the sums and the products.

$$\begin{aligned}
&\textbf{data}\ \mathsf{Tag} : \mathsf{Set}\ \textbf{where}\ \mathsf{fin}'\ \mathsf{sum}'\ \mathsf{prod}' : \mathsf{Tag}\\
&\mathsf{lang} : \mathsf{IR}\ \mathbb{N}\ \mathbb{N}\\
&\mathsf{lang} = \sigma\ \mathsf{Tag}\ \lambda\{\mathsf{fin}' \to \sigma\ \mathbb{N}\ \lambda\ n \to \iota\ n\\
&\qquad\qquad ;\mathsf{sum}' \to \delta\ 1\ \lambda\ n \to \delta\ (\mathsf{Fin}\ (n\star))\ \lambda f \to \iota\ (\mathsf{sum}\ (n\star)\ f)\\
&\qquad\qquad ;\mathsf{prod}' \to \delta\ 1\ \lambda\ n \to \delta\ (\mathsf{Fin}\ (n\star))\ \lambda f \to \iota\ (\mathsf{prod}\ (n\star)\ f)\}\\
&\mathsf{example} : \mu\ \mathsf{lang}\\
&\mathsf{example} = \mathsf{in}\ (\mathsf{sum}', (\lambda_ \to \mathsf{in}\ (\mathsf{fin}', 5, \star)), (\lambda\ n \to \mathsf{in}\ (\mathsf{fin}', n, \star)), \star)
\end{aligned}$$

The example expression denotes $\sum_{n<5} n$, and indeed, $\mathsf{decode}\ \mathsf{lang}\ \mathsf{example} = 10$.

Having introduced dependent polynomials, indexed containers and small induction recursion, we can now turn to the main focus of the paper, namely showing that they define the same class of functors and hence define the same class of data types. The key to the construction is observing that we may just as well interpret $\mathsf{IR}\ I\ O$ with our $I \to \mathsf{Set}$ presentation of slices.

$$\begin{aligned}
&[\![\cdot]\!]_{\mathsf{IR}} : \mathsf{IR}\ I\ O \to (I \to \mathsf{Set}) \to (O \to \mathsf{Set})\\
&[\![\iota\ o']\!]_{\mathsf{IR}}\ \ F\ o = o' \equiv o\\
&[\![\sigma\ S\ K]\!]_{\mathsf{IR}}\ F\ o = (s{:}S) \times ([\![K\ s]\!]_{\mathsf{IR}}\ F\ o)\\
&[\![\delta\ P\ K]\!]_{\mathsf{IR}}\ F\ o = (\mathit{if}{:}P \to \Sigma F) \times ([\![K\ (\pi_0 \circ \mathit{if})]\!]_{\mathsf{IR}}\ F\ o)
\end{aligned}$$

The correspondence up to trivial isomorphism between $[\![\cdot]\!]_{\mathsf{IR}}$ and $[\![\cdot]\!]_{\mathsf{DS}}$ is readily observed by considering F here to be an arbitrary $(X, f)^{-1}$.

4 From Poly to Small IR and Back

We divide this section into two: (*i*) we first show how to translate dependent polynomials, and hence indexed containers, into IR codes; and (*ii*) we then show how every small IR code can be translated into a dependent polynomial. Crucially, we show that these translations preserve the functorial semantics of dependent polynomials and IR codes.

From Poly to Small IR. We have already seen (example 1) that the extension of a container is an IR functor. We now extend this result to indexed containers and dependent polynomials.

Lemma 1. *Every dependent polynomial functor is an* IR *functor.*

It is enough to show that, for every dependent polynomial $(r, t, q) : \mathsf{Poly}\ I\ O$, there is an $\mathsf{IR}\ I\ O$-code, whose interpretation is isomorphic to the dependent polynomial functor $[\![(r, t, q)]\!]_{\mathsf{Poly}}$. Our candidate for this IR-code is given and interpreted as follows

$$[\![\sigma\ S\ \lambda\ s\ \to\ \delta\ ((P, t)^{-1}\ s)\ \lambda\ i\ \to\ \sigma\ (i \equiv r\ \circ\ \pi_0)\ \lambda\ \_\ \to\ \iota\ (q\ s)]\!]_{\mathsf{IR}}\ F\ o\ =\\ (s : S) \times (\mathit{if} : ((P, t)^{-1}\ s\ \to\ \Sigma F)) \times (\pi_0\ \circ\ \mathit{if} \equiv r\ \circ\ \pi_0)\ \times\ (q\ s \equiv o)$$

which is readily seen to be isomorphic to $\Sigma_q\ (\Pi_t\ (\Delta_r\ F))\ o$

$$(s : S) \times (q\ s \equiv o)\ \times\ (p : P) \to (t\ p \equiv s)\ \to\ F\ (r\ p)$$

as the former effectively constrains the function *if* to choose $r\ p$ as the index of its F, for each position $(p, \_) : (P, t)^{-1}\ s$.

From Small IR to Poly. The essence of our embedding of $\mathsf{IR}\ I\ O$ into $\mathsf{Poly}\ I\ O$ consists of showing how three constructors for $\mathsf{IR}\ I\ O$-codes can be interpreted in $\mathsf{Poly}\ I\ O$.

Definition 5. *To each code* $\gamma : \mathsf{IR}\ I\ O$ *we use structural recursion on* γ *to define a dependent polynomial* $I \leftarrow^{t_\gamma} P_\gamma \to^{r_\gamma} S_\gamma \to^{q_\gamma} O$*:*

- *if* γ *is* $\iota\ o$*, then we define* $\mathsf{S}\ \gamma = 1$*,* $\mathsf{P}\ \gamma = 0$*,* $\mathsf{r}\ \gamma = !_I$*,* $\mathsf{t}\ \gamma = !_1$*, and* $\mathsf{q}\ \gamma\ \star = o$*. As a diagram, this is as follows.* $I \leftarrow^{!} 0 \to^{!} 1 \to^{o} O$*:*
- *if* γ *is* $\sigma\ S\ K$ *then the diagram is as follows.*

$$I \xleftarrow{[\mathsf{r}\ (K\ s)]_{s:S}} \coprod_{s:S} \mathsf{P}\ (K\ s) \xrightarrow{\coprod_{s:S} \mathsf{t}\ (K\ s)} \coprod_{s:S} \mathsf{S}\ (K\ s) \xrightarrow{[\mathsf{q}\ (K\ s)]_{s:S}} O$$

  Here (and in the next clause) we use $\coprod_{s:S} m\ s$ *to abbreviate the cotuple* $[\mathsf{in}_s \circ m\ s]_{s:S}$*.*
- *if* γ *is* $\delta\ P\ K$*, the diagram is as follows.*

$$\begin{array}{ccc} \coprod_{i:P\to I} (P \times \mathsf{S}\ (K\ i)) + \mathsf{P}\ (K\ i) & \xrightarrow{\coprod_{i:P\to I}[\pi_0, \mathsf{t}\ (K\ i)]} & \coprod_{i:P\to I} \mathsf{S}\ (K\ i) \\ \downarrow{\scriptstyle [[i \circ \pi_0, \mathsf{r}\ (K\ i)]]_{i:P\to I}} & & \downarrow{\scriptstyle [\mathsf{q}\ (K\ i)]_{i:P\to I}} \\ I & & O \end{array}$$

  Note that in the last clause, it is crucial that we are dealing with small IR so that I *is a set, hence* $P \to I$ *is a set and hence the coproducts used are also small.*

We can now state the result concerning the second half of our isomorphism.

Lemma 2. *Every small* IR *functor is a dependent polynomial functor.*

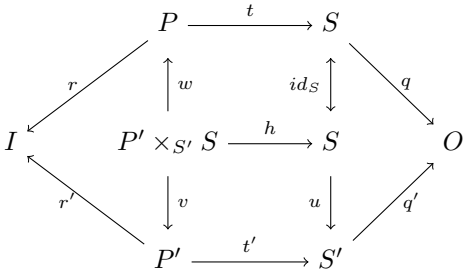
To prove the lemma we define a function $\phi : \mathsf{IR}\ I\ O \to \mathsf{Poly}\ I\ O$ by recursion on the structure of IR codes and then we prove by induction that the functorial semantics is preserved. Details of the proof can be found in the online in the expanded version of the paper at the url given in the introduction.

5 Equivalence between Small IR and Poly

We now extend our previous results to cover not just functors but also natural transformations. We will do this by (*i*) recalling the notion of morphism between dependent polynomials/indexed containers; (*ii*) introducing morphisms of IR codes, showing that the interpretation function, $[\![\_]\!]_{\mathsf{IR}} : \mathsf{IR}\ I\ O \to [\mathsf{Set}/I, \mathsf{Set}/O]$ can be extended to a functor which is full and faithful; and (*iii*) finally we prove the equivalence between the two resulting categories $\mathsf{IR}\ I\ O$ and $\mathsf{Poly}\ I\ O$. Note that our definition of morphisms for IR codes also covers the cases where I and O are large.

The Categories Poly I O and IC I O. Dependent polynomials/indexed containers with fixed input and output index sets, I and O, form a category. In this section we recall the definition of the morphisms between dependent polynomials and their interpretation as natural transformations. We conclude by stating some properties of the categories of dependent polynomials/indexed containers which allows us to recast in elementary terms the dependent polynomials introduced in definition 5.

Definition 6. *A morphism between dependent polynomials (r, t, q) and (r', t', q') is given by a diagram of the form (where the bottom square is a pullback of u and t').*



From now on, $\mathsf{Poly}\ I\ O$ will indicate the category of dependent polynomials with fixed input and output index sets I, O and their morphisms. In a similar manner we can define morphism between indexed containers.

Definition 7. *A morphism between (S, P, n) and (S', P', n') consists of*

- *a function $u : (o : O) \to S\ o \to S'\ o$;*
- *a function $f : (o : O) \to S\ o \to P'\ o\ (u\ o\ s) \to P\ o\ s$;*

such that for every $o : O$, $s : S\ o$ and $p' : P\ o\ (u\ o\ s)$ we have $n\ o\ s\ (f\ o\ s\ p') = n'\ o\ (u\ o\ s)\ p'$.

We will indicate with $\mathsf{IC}\ I\ O$ the category of indexed containers and their morphisms. The main result concerning these morphisms is the following (Theorem 2.12 in [13]). We state the result for dependent polynomials but clearly an analogue result holds also for indexed containers.

Theorem 1 ([13] Theorem 2.12). *Given dependent polynomials (r, t, q) and (r', t', q'), every natural transformation $[\![(r, t, q)]\!] \to [\![(r', t', q')]\!]$ is represented in an essentially unique way by a commuting diagram as in definition 6.*

This theorem ensures that the assignment to each dependent polynomial of its extension is a functor, and moreover this functor is full and faithful. In the following we indicate with *PolyFun I O* the full subcategory of $[\mathsf{Set}/I, \mathsf{Set}/O]$ whose objects are dependent polynomial functors and whose morphisms are natural transformation between them[3].

Corollary 1 (Representation). *For any pair of sets I, O the functor*

$$[\![\ ]\!] : \mathsf{Poly}\ I\ O \to \mathit{PolyFun}(I, O)$$

is an equivalence of categories.

Dependent polynomials and indexed containers have several interesting closure properties. Here we only need closure under set-indexed coproducts and binary product. Note that we had to define morphisms before introducing these closure properties to ensure that they have the required categorical universal properties. The sum of a K-indexed family of dependent polynomials $\{Q_k = (r_k, t_k, q_k) \mid k : K\}$, for an arbitrary set K, is the dependent polynomial $\coprod_{k:K} Q_k$ given by the following diagram

$$I \xleftarrow{[r_k]_{k:K}} \coprod_{k:K} P_k \xrightarrow{\coprod_{k:K} t_k} \coprod_{k:K} S_k \xrightarrow{[q_k]_{k:K}} O$$

where $\coprod_{k:K} t_k = [\mathsf{in}_k \circ t_k]_{k:K}$. Note that the dependent polynomial associated to $\sigma\ S\ K : \mathsf{IR}\ I\ O$ is of exactly this form. The product of two dependent polynomials (r, t, q) and (r', t', q') is the evident dependent polynomial

$$I \longleftarrow (P' \times_O S) + (P \times_O S') \longrightarrow S \times_O S' \longrightarrow O.$$

We can now describe the dependent polynomial associated to a code $\delta\ P\ K : \mathsf{IR}\ I\ O$ as the sum of products of a family of dependent polynomials. We start with a family of dependent polynomials $\{(\mathsf{r}\ (K\ i), \mathsf{t}\ (K\ i), \mathsf{q}\ (K\ i) \mid i : P \to I\}$. For each element of this family we take the product of it with the dependent polynomial

$$I \xleftarrow{i \circ \pi_0} P \times O \xrightarrow{\pi_1} O \xrightarrow{id_O} O$$

and then we take the sum of these products over the set $P \to I$.

[3] The original result for polynomial functors (Theorem 2.12 in [13]) is stated in terms of strong natural transformations. We can avoid mention of strength since natural transformations between functors on slices of Set are automatically strong.

The Category of Small IR Codes. We know how to define small IR codes and interpret them as functors between slices of Set. In this section we introduce morphisms between small IR I O-codes. Our definition will ensure that every such morphism gives rise to a natural transformation between the corresponding IR functors – and *vice versa*. We start this section developing the appropriate categorical description of the semantics of IR constructors. The constructor ι simply represents constant functors while the constructor σ takes coproducts of functors. The following lemma tells us more about the semantics of δ.

Lemma 3. *Given an object $k : X \to I$, there is a natural isomorphism*

$$[\![\delta\ P\ K]\!]\, k \cong \coprod_{i:P\to I} Hom_{\mathsf{Set}/I}(i,k) \otimes [\![K\, i]\!]_{\mathsf{IR}}\, k$$

Here $\otimes$ indicates the tensor product. Given a set X and $i : Y \to I$, the object $X \otimes i$ is nothing but the copower $\coprod_{x:X} i$, i.e the X-fold coproduct of the object i.

Proof. We have a natural isomorphism

$$\begin{aligned}[\![\delta\ P\ K]\!]\, k &= \coprod_{x:P\to X} [\![K(k\circ x)]\!]_{\mathsf{IR}}\, k \\ &\cong \coprod_{i:P\to I} \coprod_{x:P\to X} (i \equiv k\circ x) \otimes [\![K\, i]\!]_{\mathsf{IR}}\, k.\end{aligned}$$

Then observe that $\coprod_{x:P\to X}(i \equiv k\circ x) \cong Hom_{\mathsf{Set}/I}(i,k)$.

Thanks to this lemma, we are able to characterise the semantics of δ-codes through a well-known universal construction in category theory: the left Kan extension.

If $i : X \to I$ is an object in Set/I we use $(+\, i)$, in the following lemma, to indicate the functor

$$\begin{aligned}(+i) : \mathsf{Set}/I &\longrightarrow \mathsf{Set}/I \\ k &\longmapsto [i,k].\end{aligned}$$

Theorem 2. *There is a natural isomorphism*

$$[\![\delta\, P\, F]\!] \cong \coprod_{i:P\to I} Lan_{(+i)}[\![F\, i]\!]$$

Our definition of IR I O-morphisms is based on this isomorphism. First, we recall the universal property characterising the left Kan extension $Lan_G F : \mathbb{B} \to \mathbb{C}$ of a functor $F : \mathbb{A} \to \mathbb{C}$ along $G : \mathbb{A} \to \mathbb{B}$; for every functor $H : \mathbb{B} \to \mathbb{C}$ there is a bijection

$$Nat(Lan_G F, H) \cong Nat(F, H \circ G)$$

natural in H. We also need to check that IR I O-functors are closed by precomposition with functors of the form $(+i)$. Fortunately, this can be easily checked by structural induction on codes. We just state the result.

Lemma 4. *Given γ : IR I O, and a function $i : P \to I$ there exists γ^i : IR I O-code such that*

$$[\![\gamma]\!]_{\mathsf{IR}} \circ (+i) = [\![\gamma^i]\!]_{\mathsf{IR}}$$

We can now define IR morphisms by structural induction on codes as follows.

Definition 8. *Let $\gamma, \gamma' : \mathsf{IR}\ I\ O$ we define the homset $\mathsf{IR}(\gamma, \gamma')$ as follows.*
Morphisms from ι-codes:

1A. $IR(\iota\ o, \iota\ o') = o \equiv o'$
1B. $IR(\iota\ o, \sigma\ S\ K) = \coprod_{s:S} IR(\iota\ o, K\ s)$
1C. $IR(\iota\ o, \delta\ P\ K) = \coprod_{e:P\to\emptyset} IR(\iota\ o, K\ (! \circ e))$

Morphisms from σ-codes:

2. $IR(\sigma\ S\ K, \gamma) = \prod_{s:S} IR(K\ s, \gamma)$

Morphisms from δ-codes:

3. $IR(\delta\ P\ K, \gamma) = \prod_{i:P\to I} IR(K\ i, \gamma^i)$

The following theorem shows we have the right notion of morphism for IR codes.

Theorem 3. *The interpretation $[\![\_]\!]_{\mathsf{IR}}$ of $\mathsf{IR}\ I\ O$-codes can be extended to morphisms: we can associate to each $\mathsf{IR}\ I\ O$-morphism $f : \gamma \to \gamma'$ a natural transformation $[\![f]\!]_{\mathsf{IR}} : [\![\gamma]\!]_{\mathsf{IR}} \to [\![\gamma']\!]_{\mathsf{IR}}$. Moreover the following assignment is full and faithful.*

$$[\![\_]\!]_{\mathsf{IR}} : \mathsf{IR}\ I\ O\ \to\ [\mathsf{Set}/I, \mathsf{Set}/O]$$

The theorem is proved by induction on the structure of IR morphisms. Full and faithfulness allows us to reflect functor composition to the composition of small IR codes and hence we have the following important result.

Corollary 2. *$\mathsf{IR}\ I\ O$-codes and their morphisms define a category.*

An Equivalence. In the previous sections we have seen how to represent $\mathsf{IR}\ I\ O$-codes as dependent polynomials in $\mathsf{Poly}\ I\ O$ and vice versa. To sum up:

- We saw how to define a function $\psi : \mathsf{Poly}\ I\ O \to \mathsf{IR}\ I\ O$ such that $[\![\_]\!]_{\mathsf{IC}} \cong [\![\_]\!]_{\mathsf{IR}} \circ \psi$
- We saw how to define a function $\phi : \mathsf{IR}\ I\ O \to \mathsf{Poly}\ I\ O$ such that $[\![\_]\!]_{\mathsf{IR}} \cong [\![\_]\!]_{\mathsf{IC}} \circ \phi$.

We sum up these results in the following corollary.

Corollary 3. *For every $\gamma : \mathsf{IR}\ I\ O$ and, for every $(r, t, q) : \mathsf{Poly}\ I\ O$*

1) $[\![\psi \circ \phi\ (\gamma)]\!]_{\mathsf{IR}} \cong [\![\gamma]\!]_{\mathsf{IR}}$,
2) $[\![\phi \circ \psi\ (r, t, q)]\!]_{\mathsf{Poly}} \cong [\![(r, t, q)]\!]_{\mathsf{Poly}}$

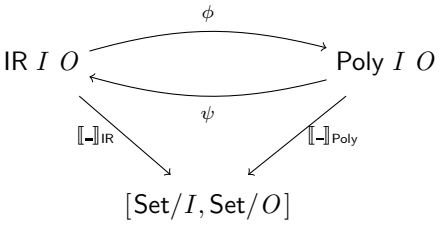
These isomorphisms deal just with objects of the two categories $\mathsf{IR}\ I\ O$ and $\mathsf{Poly}\ I\ O$. But what can we say about morphisms? As we show in the next theorem the equivalence of these two categories is an immediate consequence of the previous results combined with full and faithfulness of the respective interpretation functions:

Theorem 4. *The two categories $\mathsf{IR}\ I\ O$ and $\mathsf{Poly}\ I\ O$ are equivalent.*

It is immediate to show full and faithfulness of ϕ (or, equivalently of ψ):

$$\begin{aligned}
\mathsf{IR}\ I\ O(\gamma, \gamma') &\cong Nat([\![\gamma]\!]_{\mathsf{IR}}, [\![\gamma']\!]_{\mathsf{IR}}) && \text{(Corollary 2)}\\
&\cong Nat([\![\phi(\gamma)]\!]_{\mathsf{Poly}}, [\![\phi(\gamma')]\!]_{\mathsf{Poly}}) && \text{(Lemma 2)}\\
&\cong \mathsf{Poly}\ I\ O(\phi(\gamma), \phi(\gamma')) && \text{(Corollary 1)}
\end{aligned}$$

Now, since we have already showed that each dependent polynomial, (r, t, q) is isomorphic to $\phi(\gamma)$ for some $\gamma : \mathsf{IR}\ I\ O$ (namely $\gamma = \psi(r, t, q)$), this is enough to conclude the stated equivalence (see theorem 1, par. 4, ch. IV in [16]). Here is a commutative diagram which represents the statement of theorem 4:



6 Small Indexed Induction Recursion

The theory of induction recursion has been extended by Dybjer and Setzer in [11] in order to capture more sophisticated inductive-recursive definitions. As indexed container and dependent polynomials generalise polynomials and containers respectively, the theory of indexed induction-recursion (IIR) generalises the theory inductive-recursive definitions in order to capture not only ordinary inductive-recursive definition, but also families of inductive-recursive definitions which admit extra indexing. IR then appears as the fragment of IIR given by those definitions indexed over a singleton.

We will briefly recall the axiomatic presentation of IIR which closely follows that of IR. We then show how the theory of small indexed inductive-recursive definitions (*small* IIR) can be reduced to small IR. This simple fact will automatically transfer the results of the previous sections to small IIR, allowing to conclude a generalisation of the equivalence stated in theorem 4. We now give the coding for small IIR.

$$\begin{array}{l}
\textbf{data}\ \mathsf{IIR}\ (D : I \to \mathsf{Set})\ (E : J \to \mathsf{Set}) : \mathsf{Set1}\ \textbf{where}\\
\quad \iota : (je : \Sigma E) \to \mathsf{IIR}\ D\ E\\
\quad \sigma : (S : \mathsf{Set}) \to (K : S \to \mathsf{IIR}\ D\ E) \to \mathsf{IIR}\ D\ E\\
\quad \delta : (P : \mathsf{Set})\ (i : P \to I)\ (K : ((p : P) \to D\ (i\ p)) \to \mathsf{IIR}\ D\ E) \to \mathsf{IIR}\ D\ E
\end{array}$$

Note that δ carries an extra argument i, selecting the index for each position in P. One way to interpret these codes is by translation to the codes for $\mathsf{IR}\ \Sigma D\ \Sigma E$, as follows:

$$
\begin{array}{lcl}
\lfloor\cdot\rfloor & : & \mathsf{IIR}\ D\ E \to \mathsf{IR}\ \Sigma D\ \Sigma E \\
\lfloor \iota\ je \rfloor & = & \iota\ je \\
\lfloor \sigma\ S\ K \rfloor & = & \sigma\ S\ \lambda\ s \to \lfloor K\ s \rfloor \\
\lfloor \delta\ P\ i\ K \rfloor & = & \delta\ P\ \lambda\ iD \to \sigma\ (i \equiv (\pi_0 \circ iD))\ \lambda\ q \to \lfloor K\ (\pi_1 \circ iD) \rfloor
\end{array}
$$

In the δ case, the generated IR code yields a ΣD for each position in P, so we constrain its first component to coincide with the index required by the i in the IIR code. Given this embedding, we can endow small IIR with the categorical machinery developed for small IR. We therefore can straightforwardly define a category of $\mathsf{IIR}\ D\ E$ -codes and their morphisms. Theorem 4 in Section 5 immediately give us the following corollary.

Corollary 4. *The category* $\mathsf{IIR}\ D\ E$ *and the category* $\mathsf{Poly}\ \Sigma D\ \Sigma E$ *are equivalent.*

We can also follow Dybjer and Setzer by giving a direct interpretation of an IIR code as a functor between families of slice categories.

$$
\begin{array}{l}
[\![\cdot]\!]_{\mathsf{IIR}} : \mathsf{IIR}\ D\ E \to ((i : I) \to \mathsf{Set}/(D\ i)) \to ((j : J) \to \mathsf{Set}/(E\ j)) \\
[\![\iota\ (j', e)]\!]_{\mathsf{IIR}}\ G\ j = ((j' \equiv j), \lambda\ q \to \cdot\ q\ e) \\
[\![\sigma\ S\ K]\!]_{\mathsf{IIR}}\ G\ j = (s : S) \times ([\![K\ s]\!]_{\mathsf{IIR}}\ G\ j) \\
[\![\delta\ P\ i\ K]\!]_{\mathsf{IIR}}\ G\ j = (ig : (p : P) \to \mathsf{dom}\ (G\ (i\ p))) \times ([\![K\ (\lambda\ p \to \mathsf{fun}\ (G\ (i\ p))\ (ig\ p))]\!]_{\mathsf{IIR}}\ G\ j)
\end{array}
$$

We note that keeping I and D small ensures the following:

$$(i : I) \to \mathsf{Set}/(D\ i) \cong (i : I) \to D\ i \to \mathsf{Set} \cong \Sigma D \to \mathsf{Set} \cong \mathsf{Set}/\Sigma D$$

Consider $G\ i = (\exists.(F \circ (i,)))$ for some $F : \Sigma D \to \mathsf{Set}$ to see that $[\![\gamma]\!]_{\mathsf{IIR}}\ G$ corresponds to $[\![\lfloor\gamma\rfloor]\!]_{\mathsf{IR}}\ F$, up to bureaucratic isomorphism.

Once again, we construct simultaneously an indexed family of data types $\mu\ \gamma\ i$ and their decoders $\mathsf{decode}\ i$ as the initial algebra for $[\![\gamma]\!]_{\mathsf{IIR}}$.

$$
\begin{array}{l}
\mu\mathsf{d} : (\gamma : \mathsf{IIR}\ D\ D) \to (i : I) \to \mathsf{Set}/(D\ i) \\
\mu\mathsf{d}\ \gamma\ i = (\mu\ \gamma\ i, \mathsf{decode}\ \gamma\ i) \\
\textbf{data}\ \mu\ (\gamma : \mathsf{IIR}\ D\ D)\ (i : I) : \mathsf{Set}\ \textbf{where} \\
\quad \mathsf{in} : \mathsf{dom}\ ([\![\gamma]\!]_{\mathsf{IIR}}\ (\mu\mathsf{d}\ \gamma)\ i) \to \mu\ \gamma\ i \\
\mathsf{decode} : (\gamma : \mathsf{IIR}\ D\ D) \to (i : I) \to \mu\ \gamma\ i \to D\ i \\
\mathsf{decode}\ \gamma\ i\ (\mathsf{in}\ t) = \mathsf{fun}\ ([\![\gamma]\!]_{\mathsf{IIR}}\ (\mu\mathsf{d}\ \gamma)\ i)\ t
\end{array}
$$

The corresponding fixpoint of $[\![\lfloor\gamma\rfloor]\!]_{\mathsf{IR}}$ gives the inductive family indexed by pairs in ΣD.

Example 3. **The Bove-Capretta method, applied to call-by-value computation.** Bove and Capretta [4] make use of indexed induction-recursion to model the *domains* of partial functions. A partial function $d : (i : I) \rightharpoonup D\ i$ has a domain given by a code $\gamma : \mathsf{IIR}\ D\ D$. If $h : \mu\ \gamma\ i$ gives evidence that the domain is inhabited at argument i, then $\mathsf{decode}\ \gamma\ i\ h$ is sure to compute the result.

Let us take a concrete example. One might define a type of λ-terms and seek to give a call-by-value evaluator for them, as follows.

```
data Tm : Set where
  var : ℕ              → Tm
  app : Tm → Tm        → Tm
  lam : Tm             → Tm
```

```
cbv : Tm ⇀ Tm
cbv (var x) = var x
cbv (lam t) = lam t
cbv (app f s) with cbv f
...           | lam t = cbv (subst0 (cbv s) t)
...           | f'    = app f (cbv s)
```

where, say, we adopt a de Bruijn indexing convention and define subst0 s t to substitute s for variable *0* in t. Of course, cbv is not everywhere defined. When it is defined? It is hard to define the domain *inductively*, because the app f s case will require that subst0 (cbv s) t is in the domain whenever f is in the domain *and evaluates to* lam t. We need to define the domain simultaneously with evaluation — a job for IR.

It will prove convenient to define the special case of δ when $P = 1$.

```
δ₁ : (i : I) → (K : D i → IIR D E) → IIR D E
δ₁ i K = δ 1 (λ _ → i) λ d → K (d ⋆)
```

In the code for a domain predicate, a recursive call at i gives rise to a $\delta_1\ i\ K$ code, where K explains how to carry on if the call returns. Let us give the domain of cbv.

```
cbvD : IIR {Tm} {Tm} (λ _ → Tm) (λ _ → Tm)
cbvD = σ Tm λ
  {(var x)     → ι (var x , var x)
  ;(lam t)     → ι (lam t , lam t)
  ;(app f s)   → δ₁ f λ {(lam t) → δ₁ s λ s' → δ₁ (subst0 s' t) λ t' → ι (app f s , t')
                        ;f'      → δ₁ s λ s'                        → ι (app f s , app f' s')
  }                     }
```

Note the way the application case makes key use of the delivered values in subsequent recursive calls, and in every case, the final ι delivers an input-output pair. The type μ cbvD t thus contains the evidence that cbv t terminates without presupposing a particular value — decoding that evidence will yield t's value. The equivalence we have demonstrated in this paper ensures that the corresponding inductive *family* indexed over Tm × Tm is exactly the big-step evaluation relation for cbv.

7 Conclusion and Further Work

Despite its evident potential, the theory of induction recursion has not become as widely understood and used as it should be. In this paper we seek to broaden appreciation of Dybjer and Setzer's work by comparing it with better-known theories of data types based on dependent polynomials, and more practically with indexed containers. In the case of *small* IR, these three analyses coincide. We can now pick up the fruits of our central result (theorem 4).

Initial Algebras. When interpreting codes in $\mathsf{IR}\ I\ I$ we get endofunctors on Set/I. Theorem 4 ensures that initial algebras for these functors always exist, since they are initial algebras for dependent polynomial endofunctors. Altenkirch and Morris have [2] given parametrized initial algebras of indexed containers of type $\mathsf{IC}\ (I + O)\ O$: as a result of this work, the same construction carries over into $\mathsf{IR}\ (I + O)\ O$ functors. We also now know that functors definable by Small IR also have final coalgebras - these are just the final coalgebras for dependent polynomial functors/indexed containers. They have recently investigated by Capretta in unpublished work under the name of Wander types.

Closure Properties. The axiomatization of small IR and its semantics provides a new (but equivalent) grammar to work with the categories Poly and IC. It is known that these categories have very rich closure properties such as sums, products, composition, as well as linear and differential structure. Clearly we can transport these properties along the equivalence of theorem 4.

Compositions. A difficult open question in the theory of induction-recursion is whether the Dybjer-Setzer functors are closed under composition: given codes $\gamma : \mathsf{IR}\ I\ J$ and $\gamma' : \mathsf{IR}\ J\ O$ is it always possible to find a code ξ in $\mathsf{IR}\ I\ O$ such that $[\![\gamma']\!] \circ [\![\gamma]\!] \cong [\![\xi]\!]$? Theorem 4 ensures that we can transport composition in Poly or IC to obtain closure under composition of small IR functors.

Further Work. We have proved that $\mathsf{Poly}\ I\ O$, $\mathsf{IC}\ I\ O$ and $\mathsf{IR}\ I\ O$ are equivalent categories which define the same class of functors. It is easy to generalize this result to a biequivalence of bicategories. Since it is possible to define reindexing of IR codes and IR functors, in future work we would like to explore this extra-structure of small IR and compare it with the double category of Poly. Abstracting from the category of sets we also aim to investigate to which extent this result applies to arbitrary LCCCs.

References

1. Abbott, M., Altenkirch, T., Ghani, N.: Containers. Constructing Strictly Positive Types. TCS 342, 3–27 (2005)
2. Altenkirch, T., Morris, P.: Indexed containers. In: Procs. of the 24th Annual IEEE Symposium on Logic in Computer Science (LICS 2009). IEEE Computer Society (2009)
3. Aczel, P.: An introduction to inductive definition. In: Barwise, J. (ed.) Handbook of Mathematical Logic, pp. 739–782. North-Holland, Amsterdam (1977)
4. Bove, A., Capretta, V.: Nested General Recursion and Partiality in Type Theory. In: Boulton, R.J., Jackson, P.B. (eds.) TPHOLs 2001. LNCS, vol. 2152, pp. 121–135. Springer, Heidelberg (2001)
5. Clairambault, P., Dybjer, P.: The Biequivalence of Locally Cartesian Closed Category and Martin Löf Type Theories, arXiv:1112.3456v1 [cs.LO], December 15 (2011)
6. Coquand, T., Dybjer, P.: Inductive Definitions and Type Theory an Introduction. In: Thiagarajan, P.S. (ed.) FSTTCS 1994. LNCS, vol. 880, pp. 60–76. Springer, Heidelberg (1994)

7. Curien, P.-L.: Substitution up to isomorphism. Fundamenta Informaticae 19(1-2), 51–86 (1993)
8. Dybjer, P.: A general formulation of simultaneous inductive-recursive definitions in type theory. Journal of Symbolic Logic 65(2), 525–549 (2000)
9. Dybjer, P., Setzer, A.: A Finite Axiomatization of Inductive-Recursive Definitions. In: Girard, J.-Y. (ed.) TLCA 1999. LNCS, vol. 1581, pp. 129–146. Springer, Heidelberg (1999)
10. Dybjer, P., Setzer, A.: Induction-recursion and initial algebras. Annales of Pure and Applied Logic 124, 1–47 (2003)
11. Dybjer, P., Setzer, A.: Indexed Induction-Recursion. Journal of Logic and Algebraic Programming 66(1), 1–49 (2006)
12. Gambino, N., Hyland, M.: Wellfounded trees and dependent polynomial functors. In: Berardi, S., Coppo, M., Damiani, F. (eds.) TYPES 2003. LNCS, vol. 3085, pp. 210–225. Springer, Heidelberg (2004)
13. Gambino, N., Kock, J.: Polynomial functors and polynomial monads arXiv:0906.4931v2 [math.CT], March 6 (2010)
14. Hofmann, M.: On the interpretation of type theory in locally cartesian closed categories. In: Pacholski, L., Tiuryn, J. (eds.) CSL 1994. LNCS, vol. 933, pp. 427–441. Springer, Heidelberg (1995)
15. Kock, J.: Notes on Polynomial functors, `http://www.mat.uab.es/~kock/cat/polynomial.html`
16. Mac Lane, S.: Categories for the working mathematician, 2nd edn. Springer, New York (1998)
17. Martin-Löf, P.: An intuitionistic theory of types: Predicative part. In: Logic Colloquium 1973, pp. 73–118. North-Holland, Amsterdam (1973)
18. Moerdijk, I., Palmgren, E.: Wellfounded trees in categories. Annals of Pure and Applied Logic 104, 189–218 (2000)
19. Seely, R.A.G.: Locally cartesian closed categories and type theory. Math. Proc. Cambridge Philos. Soc. (95), 33–48 (1984)