Variations on Inductive-Recursive Definitions

Neil Ghani[1], Conor McBride[1], Fredrik Nordvall Forsberg[1], and Stephan Spahn[2]

1 **University of Strathclyde, Glasgow, Scotland**
2 **Middlesex University, London, England**

Abstract

Dybjer and Setzer introduced the definitional principle of inductive-recursively defined families — i.e. of families $(\mathsf{U} : \mathsf{Set}, \mathsf{T} : \mathsf{U} \to D)$ such that the inductive definition of U may depend on the recursively defined T — by defining a type $\mathsf{DS}\ D\ E$ of codes. Each $c : \mathsf{DS}\ D\ E$ defines a functor $[\![\ c\ ]\!] : \mathsf{Fam}\ D \to \mathsf{Fam}\ E$, and $(\mathsf{U}, \mathsf{T}) = \mu[\![\ c\ ]\!] : \mathsf{Fam}\ D$ is exhibited as the initial algebra of $[\![\ c\ ]\!]$.

This paper considers the composition of DS-definable functors: Given $F : \mathsf{Fam}\ C \to \mathsf{Fam}\ D$ and $G : \mathsf{Fam}\ D \to \mathsf{Fam}\ E$, is $G \circ F : \mathsf{Fam}\ C \to \mathsf{Fam}\ E$ DS-definable, if F and G are? We show that this is the case if and only if powers of families are DS-definable, which seems unlikely. To construct composition, we present two new systems UF and PN of codes for inductive-recursive definitions, with $\mathsf{UF} \hookrightarrow \mathsf{DS} \hookrightarrow \mathsf{PN}$. Both UF and PN are closed under composition. Since PN defines a potentially larger class of functors, we show that there is a model where initial algebras of PN-functors exist by adapting Dybjer-Setzer's proof for DS.

1998 ACM Subject Classification F.3.3, F.4.1.

Keywords and phrases Type Theory, induction-recursion, initial-algebra semantics.

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.63

1 Introduction

Codes for inductive-recursive definitions were introduced in a series of papers by Dybjer and Setzer [6, 7, 8]. An initial motivation [5] was to give generic rules that can be specialised to define most types occurring in Martin-Löf Type Theory [13], including inductive families [4] and Tarski-style universes [14]. An inductive-recursive definition defines not only a type, but more generally a family $(\mathsf{U} : \mathsf{Set}, \mathsf{T} : \mathsf{U} \to D)$ of types for some $D : \mathsf{Set}_1$, where the inductive definition of U may depend on the recursively defined T; examples can be found in Section 2. To represent such definitions, Dybjer and Setzer introduced a type $\mathsf{DS}\ D\ E$ of codes representing functors $\mathsf{Fam}\ D \to \mathsf{Fam}\ E$. The family $(\mathsf{U}, \mathsf{T}) = \mu[\![\ c\ ]\!] : \mathsf{Fam}\ D$ arises as the initial algebra of a functor $[\![\ c\ ]\!] : \mathsf{Fam}\ D \to \mathsf{Fam}\ D$ represented by a code $c : \mathsf{DS}\ D\ D$.

Induction-recursion is important as it is the strongest form of inductive definition we have, surpassing, for example, inductive definitions [10] and inductive families [2]. This paper asks the following fundamental and significant question:

Is the theory of inductive-recursive definitions, as currently understood, optimal?

We still believe that conceiving of inductive-recursive definitions as initial algebras in the category $\mathsf{Fam}\ D$ is the right thing to do. However, the current type of codes for generating such functors may not actually be optimal for this purpose. We come to this conclusion by considering the question of composition of codes. Given $[\![\ c\ ]\!] : \mathsf{Fam}\ C \to \mathsf{Fam}\ D$ and $[\![\ d\ ]\!] : \mathsf{Fam}\ D \to \mathsf{Fam}\ E$ represented by Dybjer-Setzer codes $c : \mathsf{DS}\ C\ D$ and $d : \mathsf{DS}\ D\ E$ respectively, is $[\![\ d\ ]\!] \circ [\![\ c\ ]\!] : \mathsf{Fam}\ C \to \mathsf{Fam}\ E$ DS-definable, i.e. is there a code $d \bullet c : \mathsf{DS}\ C\ E$

such that $[\![\ d \bullet c\ ]\!] = [\![\ d\ ]\!] \circ [\![\ c\ ]\!]$? A positive answer would allow modularity in datatype definitions, as one can then replace all inductive arguments (U, T) in a datatype by $F(U, T)$ for any DS-definable functor F by composing with the code for F. For instance, a code for an inductive definition of multiway trees, where each node has a list of subtrees, can be constructed by composing a code for lists with itself. Other classes of data types such as inductive definitions or inductive families are closed under composition [10] — it is a property naturally to be expected from the viewpoint of initial algebra semantics.

It is currently unknown whether DS is closed under composition, although we suspect that it is not. In support of this claim, we show that DS is closed under composition *if and only if* powers $A \to [\![\ c\ ]\!]$ of codes are definable (Section 2, where we also recall Dybjer and Setzer's codes). Since such a power operation is unlikely to exist, we are led to investigate alternative systems of inductive-recursive definitions that *are* closed under composition.

We first introduce a system UF of *uniform* codes for inductive-recursive definitions, and their decoding (Section 3). This system can be regarded as a subsystem of DS, and as such, it is clear that UF-functors have initial algebras since DS-functors do. The novum is that uniformity of codes can be exploited to define powers, which in turn means that a composition operator for uniform codes is obtainable, i.e. we have isolated a subclass of DS-functors that is closed under composition. Next we introduce another system PN of *polynomial* codes, and their decoding (Section 4). Notably, PN contains a constructor for the dependent product of codes which ensures that PN is closed under composition. PN is a supersystem of DS and hence we cannot inherit initial algebras for PN from DS, but must prove their existence directly: we do so by adapting Dybjer-Setzer's proof to our more general setting. The introduction of new alternative formulations of induction-recursion has the potential to have significant impact if they — as we believe to be the case — have better properties than the current one and hence come to supplant the current formulation.

Type-theoretic notation and assumptions We work informally in a standard type theory with dependent function spaces $\big(x : A\big) \to B(x)$ (written $A \to B$ if x does not occur in B), dependent pair types $\big(\Sigma x : A\big)B(x)$ (written $A \times B$ if x does not occur in B), coproducts $A + B$ with injections inl and inr, and an identity type which we shall simply write as $a = b$. Finite enumerations are denoted by $\{a_1, a_2, \ldots, a_m\}$; instances include $\mathsf{0} = \{\}$, $\mathsf{1} = \{\star\}$ and $\mathsf{2} = \{\mathsf{ff}, \mathsf{tt}\}$. We write anonymous functions as $(a \mapsto b)$, or $(\_ \mapsto b)$ when the argument is not used by the function. We assume two universes à la Russell $\mathsf{Set} : \mathsf{Set}_1$, with $A : \mathsf{Set}$ implying $A : \mathsf{Set}_1$. We assume function extensionality, i.e. that pointwise equal functions are equal. This is essential in our development. For simplicity, we also assume uniqueness of identity proofs, i.e. that if $p : a = b$ and $q : a = b$, then $p = q$, but this assumption should be avoidable with a little more work. In any case, both of these assumptions are valid in extensional Type Theory [14], which readily has a set-theoretic interpretation. The content of this paper (except for the set-theoretical model of PN) has been formalised in Agda[1].

2 Dybjer-Setzer Codes DS for Inductive-Recursive Definitions

We recall the system of Dybjer-Setzer codes DS, how codes represent inductive-recursive definitions, and finally prove powers to be necessary and sufficient for DS to be closed under composition.

[1] Available at `http://personal.cis.strath.ac.uk/fredrik.nordvall-forsberg/variantsIR/`.

2.1 Definition of DS and its Decoding

For $D,\ E : \mathsf{Set}_1$, the type $\mathsf{DS}\ D\ E$ consists of codes that represent functors $\mathsf{Fam}\ D \to \mathsf{Fam}\ E$ describing the constructors of inductive-recursive definitions.

▶ **Definition 1.** Given $D, E : \mathsf{Set}_1$, the large type $\mathsf{DS}\ D\ E : \mathsf{Set}_1$ of *Dybjer-Setzer codes* is inductively defined by the following generators:

$$\iota : E \to \mathsf{DS}\ D\ E$$
$$\sigma : \big(A : \mathsf{Set}\big) \to (A \to \mathsf{DS}\ D\ E) \to \mathsf{DS}\ D\ E$$
$$\delta : \big(A : \mathsf{Set}\big) \to ((A \to D) \to \mathsf{DS}\ D\ E) \to \mathsf{DS}\ D\ E$$

Here ι shall represent trivial functors, σ sums of functors, and δ dependent sums. See Dybjer and Setzer [7] for a more in-depth explanation, and the examples below for intuition. Note that Dybjer and Setzer only considered systems of the form $\mathsf{DS}\ D\ D$, i.e. where $E = D$. For our purposes the more general formulation will be clearer; it also accounts for the fact that $\mathsf{DS}\ D\ E$ is functorial covariantly in E and contravariantly in D.

▶ **Example 2** (W-types)**.** By choosing $D = E = 1$, we can use $\mathsf{DS}\ 1\ 1$ to represent inductive definitions. Let us encode Martin-Löf's type $\mathsf{W}\ S\ P : \mathsf{Set}$ of wellfounded trees, where $S : \mathsf{Set}$ encodes the possible shapes of the tree, and $P : S \to \mathsf{Set}$ maps each shape to its branching degree. This type is inductively defined by the constructor

$$\mathsf{sup} : \big(s : S\big) \to (P(s) \to \mathsf{W}\ S\ P) \to \mathsf{W}\ S\ P$$

Here we see that sup takes one non-inductive argument $s : S$, followed by an inductive argument $P(s) \to \mathsf{W}\ S\ P$, which depends on the first non-inductive one. We will see shortly in Example 5 that $\mathsf{W}\ S\ P$ can be represented by the code $c_{\mathsf{W}\ S\ P} : \mathsf{DS}\ 1\ 1$ with $c_{\mathsf{W}\ S\ P} = \sigma\, S\, (s \mapsto \delta\, P(s)\, (\_ \mapsto \iota\star))$ where σ is used for the non-inductive argument and δ for the inductive one, finally finishing off with a trivial ι.

▶ **Example 3** (A universe closed under W-types)**.** We get considerably more power by choosing $D = E = \mathsf{Set}$. Now we can represent a universe containing 2 that is *closed under* W*-types* by the code $c_{2\mathsf{W}} : \mathsf{DS}\ \mathsf{Set}\ \mathsf{Set}$, where

$$c_{2\mathsf{W}} = \sigma\ \{\mathsf{two}, \mathsf{w}\}\ (\mathsf{two} \mapsto \iota\ 2; \mathsf{w} \mapsto \delta\ 1\ (X \mapsto (\delta\ (X\star)\ (Y \mapsto \iota\ (\mathsf{W}\ (X\star)\ Y)))))$$

First we offer a choice between two constructors: two and w using σ. In the two case, we use an ι code to ensure the name two decodes to 2; in the w case, we ask for a name s for the shapes of the W-type using $\delta\ 1$, and for every element in the decoding of that name, we ask for a name for the branching degrees using $\delta\ (X\star)$ — here $X : 1 \to \mathsf{Set}$ represents the decoding of the name s. The rest of the code gets to depend on the decoding $Y : X\star \to \mathsf{Set}$ of this family, and we finish by declaring that this constructor decodes to $\mathsf{W}\ (X\star)\ Y$. Note that this code can be written as a coproduct of codes $c_2\ +_{\mathsf{DS}}\ c_{\mathsf{W}}$: generally for $c\ d : \mathsf{DS}\ D\ E$, we define their coproduct $c +_{\mathsf{DS}} d = \sigma\ 2\ (\mathsf{ff} \mapsto c\ ;\ \mathsf{tt} \mapsto d)$. We will return to this in Example 11.

Decoding of Dybjer-Setzer codes as functors on families make the above intuitions precise. For $D : \mathsf{Set}_1$, $\mathsf{Fam}\ D$ is the category where objects are families of Ds, i.e. pairs (A, P) where $A : \mathsf{Set}$ and $P : A \to D$; a morphism $(A, P) \to (B, Q)$ consists of a function $f : A \to B$ together with a proof that $Q(f(a)) = P(a)$ for each $a : A$. For future reference, we note that Fam is a functor with action on morphisms $\mathsf{Fam}(h)\,(A, P) = (A, h \circ P)$ and moreover a monad with unit $\eta_{\mathsf{Fam}}(e) = (1, \_ \mapsto e)$ and multiplication $\mu_{\mathsf{Fam}} : \mathsf{Fam}\ (\mathsf{Fam}\ D) \to \mathsf{Fam}\ D$ given by $\mu_{\mathsf{Fam}}(A, P) = \big((\Sigma x : A)(P(x)_0), (x, y) \mapsto (P(x))_1\ y\big)$ where we have written $P(x)_0$ and $P(x)_1$ for the components of the family $P(x) = (P(x)_0, P(x)_1)$.

▶ **Definition 4.** Let $D, E : \mathsf{Set}_1$ and $c : \mathsf{DS}\ D\ E$. We define the *decoding* of c as the functor $[\![\ c\ ]\!] : \mathsf{Fam}\ D \to \mathsf{Fam}\ E$ given by $[\![\ c\ ]\!](A,P) = ([\![\ c\ ]\!]_0(A,P), [\![\ c\ ]\!]_1(A,P))$, where $[\![\ \_\ ]\!]_0 : \mathsf{DS}\ D\ E \to \mathsf{Fam}\ D \to \mathsf{Set}$ and $[\![\ \_\ ]\!]_1 : \big(c : \mathsf{DS}\ D\ E\big) \to \big(Z : \mathsf{Fam}\ D\big) \to [\![\ c\ ]\!]_0\ Z \to E$ are defined by

$$
\begin{aligned}
&[\![\ \iota\ e\ ]\!]_0\ (U,T) = 1 && [\![\ \iota\ e\ ]\!]_1\ (U,T)\ \star = e\\
&[\![\ \sigma\ A\ f\ ]\!]_0\ (U,T) = \big(\Sigma a : A\big)([\![\ f\ a\ ]\!]_0\ (U,T)) && [\![\ \sigma\ A\ f\ ]\!]_1\ (U,T)\ (a,x) = [\![\ f\ a\ ]\!]_1\ (U,T)\ x\\
&[\![\ \delta\ A\ F\ ]\!]_0\ (U,T) = && [\![\ \delta\ A\ F\ ]\!]_1\ (U,T)\ (g,x) =\\
&\quad \big(\Sigma g : A \to U\big)([\![\ F\ (T \circ g)\ ]\!]_0\ (U,T)) && \qquad [\![\ F\ (T \circ g)\ ]\!]_1\ (U,T)\ x
\end{aligned}
$$

▶ **Example 5.** For decoding Example 2, note that $\mathsf{Fam}\ 1 \cong \mathsf{Set}$ since the second component of such a family is trivial. Thus, if $(W,T) : \mathsf{Fam}\ 1$, then

$$[\![\ c_{\mathsf{W}\ S\ P}\ ]\!]_0(W,T) = \big(\Sigma s : S\big)\big((P(s) \to W) \times 1\big) \tag{1}$$

such that indeed $\mathsf{sup} : [\![\ c_{\mathsf{W}\ S\ P}\ ]\!]_0(\mathsf{W}\ S\ P, \_) \to \mathsf{W}\ S\ P$ (up to isomorphism), and initial algebras of $[\![\ c_{\mathsf{W}\ S\ P}\ ]\!] : \mathsf{Fam}\ 1 \to \mathsf{Fam}\ 1$ are W-types. Instead of leaving the fibres of the family trivial, we can "upgrade" the given code to do something interesting in the whole family. For instance, if we redefine $c_{\mathsf{W}\ S\ P} : \mathsf{DS}\ \mathsf{Set}\ \mathsf{Set}$ by

$$c_{\mathsf{W}\ S\ P} = \sigma\ S\ (s \mapsto \delta\ P(s)\ (Y \mapsto \iota\ \Big(\big(x : P(s)\big) \to Y\ x\Big)))$$

the index type decoding (1) stays the same, but the decoding $[\![\ c_{\mathsf{W}\ S\ P}\ ]\!]_1(W,T)$ applies T everywhere in the given structure. In particular, if we choose $S = \mathbb{N}$ and $P = \mathsf{Fin}$, where $\mathsf{Fin}\ n$ is a finite type with n elements, then $[\![\ c_{\mathsf{W}\ \mathbb{N}\ \mathsf{Fin}}\ ]\!](W,T) \cong (\mathsf{List}\, W,\ [w_1, \ldots, w_n] \mapsto T\, w_1 \times \ldots \times T\, w_n)$. We will see a use of this upgraded code later in Example 21.

▶ **Example 6.** Similarly, the decoding of the code $c_{2\mathsf{W}} : \mathsf{DS}\ \mathsf{Set}\ \mathsf{Set}$ from Example 3 satisfies $[\![\ c_{2\mathsf{W}}\ ]\!]_0(U,T) \cong 1 + \big(\Sigma s : U\big)(T(s) \to U)$ with $[\![\ c_{2\mathsf{W}}\ ]\!]_1(U,T)\ (\mathsf{inl}\star) = 2$ and $[\![\ c_{2\mathsf{W}}\ ]\!]_1(U,T)\ (\mathsf{inr}\,(s,p)) = \mathsf{W}\ (T\,s)\ (T \circ p)$ which are the equations for a universe closed under W-types.

Dybjer and Setzer [7] also give rules ensuring that $[\![\ c\ ]\!] : \mathsf{Fam}\ D \to \mathsf{Fam}\ D$ has an initial algebra $(\mathsf{U}_{[\![\ c\ ]\!]}, \mathsf{T}_{[\![\ c\ ]\!]})$ for every $c : \mathsf{DS}\ D\ D$. We omit them here.

2.2 Composition of DS codes

We are now approaching the actual topic of the paper. Given DS-codes $c : \mathsf{DS}\ C\ D$ and $d : \mathsf{DS}\ D\ E$, is there a code $d \bullet c : \mathsf{DS}\ C\ E$ such that $[\![\ d \bullet c\ ]\!](U,T) \cong [\![\ d\ ]\!]([\![\ c\ ]\!](U,T))$? We immediately notice that it is easy to define postcomposition of any code by a ι or a σ code: the functor $[\![\ \iota\ e\ ]\!]$ ignores its argument, hence so must $[\![\ (\iota\ e) \bullet c\ ]\!]$, and for σ codes, we can just proceed structurally. The δ case, however, requires more thought. Again, looking first at the action on index types of the families, we find for the right hand side of the above equation

$$
\begin{aligned}
[\![\ \delta\ A\ F\ ]\!]_0([\![\ c\ ]\!]_0 Z) &= \big(\Sigma g : A \to [\![\ c\ ]\!]_0 Z\big)\big([\![\ F\big([\![\ c\ ]\!]_1(Z) \circ g\big)\ ]\!]_0([\![\ c\ ]\!]Z)\big)\\
&= \Big(\big(A \longrightarrow_{\mathsf{Fam}} [\![\ c\ ]\!]Z\big) \gg\!=_{\mathsf{Fam}} \big(g \mapsto [\![\ F\big([\![\ c\ ]\!]_1(Z) \circ g\big)\ ]\!]_0([\![\ c\ ]\!]Z)\big)\Big)_0
\end{aligned}
$$

where $\_ \gg\!=_{\mathsf{Fam}} \_ : \mathsf{Fam}\ D \to (D \to \mathsf{Fam}\ E) \to \mathsf{Fam}\ E$ is the bind of the Fam monad defined by $Z \gg\!=_{\mathsf{Fam}} h = \mu_{\mathsf{Fam}}\,(\mathsf{Fam}(h)\,Z)$, and

$$
\begin{aligned}
&\_ \longrightarrow_{\mathsf{Fam}} \_ : \big(S : \mathsf{Set}\big) \to \mathsf{Fam}\ D \to \mathsf{Fam}\ (S \to D)\\
&S \longrightarrow_{\mathsf{Fam}} (A,P) = (S \to A, g \mapsto P \circ g)
\end{aligned}
$$

is a power in the category of elements $\big(\Sigma D : \mathsf{Set}_1\big)(\mathsf{Fam}\ D)$ of the functor Fam. This suggests that to define $(\delta\ A\ F) \bullet c$, we need to internalise $\gg\!=_{\mathsf{Fam}}$ and $\longrightarrow_{\mathsf{Fam}}$ in the system DS. The first is readily achievable, because $\mathsf{DS}\ C$ is also a monad [11]:

▶ **Proposition 7.** *There is an operation* $\_ \gg\!= \_ : \mathsf{DS}\ C\ D \to (D \to \mathsf{DS}\ C\ E) \to \mathsf{DS}\ C\ E$ *such that* $[\![\ c \gg\!= g\ ]\!]\ Z \cong [\![\ c\ ]\!]\ Z \gg\!=_{\mathsf{Fam}} (e \mapsto [\![\ g\, e\ ]\!]\ Z)$ *for every* $Z : \mathsf{Fam}\ C$, $c : \mathsf{DS}\ C\ D$ *and* $g : D \to \mathsf{DS}\ C\ E$. ◀

Thus it remains to define powers of codes. Here, however, we hit a wall trying to define $S \longrightarrow c$ by induction on c: to apply the inductive hypothesis on $f\ a$ in the following S-fold power of a σ code

$$S \to [\![\ \sigma\ A\ f\ ]\!]_0\ Z = S \to \big(\Sigma a : A\big)([\![\ f\ a\ ]\!]_0\ Z) \cong \big(\Sigma g : S \to A\big)\big((x : S) \to [\![\ f\ (g\ x)\ ]\!]_0\ Z\big)$$

we would need to generalise our construction to *dependent products* $\big(x : S\big) \to c(x)$ where $c : S \to \mathsf{DS}\ D\ E$. But, if we do so, we can no longer do an induction on c, and we are stuck. Even worse, any definition of composition necessarily involves powers:

▶ **Theorem 8.** *There is a composition operator for* DS *if and only if there is a power operator for* DS. *Here, by composition and power operators we mean terms*

$$\_ \bullet \_ : \mathsf{DS}\ D\ E \to \mathsf{DS}\ C\ D \to \mathsf{DS}\ C\ E$$
$$\_ \longrightarrow \_ : \big(S : \mathsf{Set}\big) \to \mathsf{DS}\ D\ E \to \mathsf{DS}\ D\ (S \to E)$$

respectively such that $[\![\ c \bullet d\ ]\!]\ Z \cong [\![\ c\ ]\!]([\![\ d\ ]\!]\ Z)$ *and* $[\![\ S \longrightarrow c\ ]\!]\ Z \cong (S \longrightarrow_{\mathsf{Fam}} [\![\ c\ ]\!]\ Z)$.

Proof. Given $\_ \longrightarrow \_$, we can define $\_ \bullet \_$ by

$$\begin{aligned}(\iota\ e) \bullet d &= \iota\ e\\ (\sigma\ A\ f) \bullet d &= \sigma\ A\ (a \mapsto (f\ a) \bullet d)\\ (\delta\ A\ F) \bullet d &= (A \longrightarrow d) \gg\!= (g \mapsto (F\ g) \bullet d)\end{aligned}$$

using Proposition 7. Conversely, $A \longrightarrow c \coloneqq (\delta\ A\ (h \mapsto \iota\ h)) \bullet c$ is a power operator. ◀

Two natural options suggest themselves as solutions: (i) restrict codes to ensure that no dependency arises in the definition of powers; (ii) devise a system with dependent products of codes. In the next two sections, we investigate new systems of codes for both of these solutions.

3 Uniform Codes UF for Inductive-Recursive Definitions

This section presents our first new system for induction-recursion with a native composition operation. The system UF of *uniform codes* is a subsystem of DS (Proposition 14). Informally, a uniform code is a DS code where, for every constructor in a term, all immediate subterms have the same root-constructor. Thus (the shape of) $\sigma\ A\ (a \mapsto \delta\ B(a)\ (h \mapsto \iota\ \phi(a,h)))$ is uniform, whereas $\sigma\ A\ f +_{\mathsf{DS}} \delta\ B\ G = \sigma\ 2\ (\mathsf{tt} \mapsto \sigma\ A\ f; \mathsf{ff} \mapsto \delta\ B\ G)$ is not since one subcode is a σ code while the other is a δ one. Uniform codes originated with Peter Hancock [12].

3.1 Definition of UF and its Decoding

Formally, we define a type of codes $\mathsf{Uni}\ D : \mathsf{Set}_1$ determining the code shapes, simultaneously with a function $\mathsf{Info} : \mathsf{Uni}\ D \to \mathsf{Set}_1$, which assigns to each code the information available for indexing codes depending on it, in a uniform way.

▶ **Definition 9.** Let $D, E : \mathsf{Set}_1$. The large type $\mathsf{UF}\ D\ E : \mathsf{Set}_1$ of *uniform codes for induction-recursion* is defined by $\mathsf{UF}\ D\ E \coloneqq \big(\Sigma c : \mathsf{Uni}\ D\big)(\mathsf{Info}\ c \to E)$, where $\mathsf{Uni}\ D : \mathsf{Set}_1$ and $\mathsf{Info} : \mathsf{Uni}\ D \to \mathsf{Set}_1$ are mutually defined by

$$\begin{aligned}
&\iota_{\mathsf{UF}} : \mathsf{Uni}\ D && \mathsf{Info}\ \iota_{\mathsf{UF}} = 1\\
&\sigma_{\mathsf{UF}} : \big(c : \mathsf{Uni}\ D\big) \to (\mathsf{Info}\ c \to \mathsf{Set}) \to \mathsf{Uni}\ D && \mathsf{Info}\ (\sigma_{\mathsf{UF}}\ c\ A) = \big(\Sigma\gamma : \mathsf{Info}\ c\big)(A\ \gamma)\\
&\delta_{\mathsf{UF}} : \big(c : \mathsf{Uni}\ D\big) \to (\mathsf{Info}\ c \to \mathsf{Set}) \to \mathsf{Uni}\ D && \mathsf{Info}\ (\delta_{\mathsf{UF}}\ c\ A) = \big(\Sigma\gamma : \mathsf{Info}\ c\big)(A\ \gamma \to D)
\end{aligned}$$

It is easy to see that $\mathsf{UF}\ D\ \_$ is functorial by function composition (alternatively, it is defined as the action of a container [1], and hence automatically functorial). This two-level presentation of codes $(\mathsf{Uni}, \mathsf{Info})$ has similarities with the $(\mathsf{SP}, \mathsf{Arg})$ presentation of Dybjer-Setzer codes in the original paper [6], where however SP was merely inductively defined, whereas here $(\mathsf{Uni}, \mathsf{Info})$ is itself an inductive-recursive definition. A further difference is that the definition of Uni is left-nested while SP as well as DS are right-nested in the sense of Pollack [15]. This can be seen as the source of uniformity in the definition.

▶ **Example 10** (W-types, again)**.** In order to get a feel for uniform codes, we return to the W-types of Example 2. A uniform code in $\mathsf{UF}\ 1\ 1$ representing the W-type $\mathsf{W}\ S\ P$ is $c_{\mathsf{W}\ S\ P,\mathsf{UF}} = \delta_{\mathsf{UF}}\ \big(\sigma_{\mathsf{UF}}\ \iota_{\mathsf{UF}}\ (\_ \mapsto S)\big)\ ((\_, s) \mapsto P(s)) : \mathsf{Uni}\ 1$, together with the terminal map $\mathsf{Info}\ c_{\mathsf{W}\ S\ P,\mathsf{UF}} \to 1$. If we compare this to the Dybjer-Setzer code from Example 2, we see that the order of the (non-base-case) constructors is reversed:

$$\begin{aligned}
(\delta_{\mathsf{UF}}\ \big(\sigma_{\mathsf{UF}}\ \iota_{\mathsf{UF}}\ (_ \mapsto S)\big)\ ((_, s) \mapsto P(s))\ ,\ _ \mapsto \star) &: \mathsf{UF}\ 1\ 1\\
\sigma\ S\ (s \mapsto \delta\ P(s)\ (_ \mapsto \iota\star)) &: \mathsf{DS}\ 1\ 1
\end{aligned}$$

Also this code can be "upgraded" to a more interesting $\mathsf{UF}\ \mathsf{Set}\ \mathsf{Set}$ code applying a given T everywhere. We get the same decoding as in Example 5 if we replace the trivial map $(\_ \mapsto \star) : \mathsf{Info}\ c_{\mathsf{W}\ S\ P,\mathsf{UF}} \to 1$ by the map $(s, Y, \star) \mapsto \big(x : P(s)\big) \to Y\ x$.

▶ **Example 11** (A universe closed under W-types, again)**.** Example 3 uses coproducts of DS codes. Coproducts of uniform codes a priori do not always exist as the different summands may have different shapes. However, we will prove coproducts of uniform codes to exist in Section 3.3. Assuming, for now, the coproduct $\_ +_{\mathsf{UF}} \_ : \mathsf{UF}\ D\ E \to \mathsf{UF}\ D\ E \to \mathsf{UF}\ D\ E$, we construct the code $c_{2,\mathsf{UF}} +_{\mathsf{UF}} c_{\mathsf{W},\mathsf{UF}} : \mathsf{UF}\ \mathsf{Set}\ \mathsf{Set}$ from the following summands — again note that the nesting is the other way around compared to the DS code in Example 3:

$$\begin{aligned}
c_{2,\mathsf{UF}} &= (\iota_{\mathsf{UF}}, \star \mapsto 2) : \mathsf{UF}\ \mathsf{Set}\ \mathsf{Set}\\
c_{\mathsf{W},\mathsf{UF}} &= \big(\delta_{\mathsf{UF}}\ \big(\delta_{\mathsf{UF}}\ \iota_{\mathsf{UF}}\ (\star \mapsto 1)\big)\ ((\star, S) \mapsto S\star), ((\star, S), P) \mapsto \mathsf{W}\ (S\star)\ P\big) : \mathsf{UF}\ \mathsf{Set}\ \mathsf{Set}
\end{aligned}$$

Decoding of uniform codes $\mathsf{UF}\ D\ E$ is again given by functors $\mathsf{Fam}\ D \to \mathsf{Fam}\ E$. The definition is very similar to the decoding of DS codes except that UF codes have two components. We use the same notation $[\![-]\!]$ for decoding a uniform code as for decoding a DS code; this convention is reasonable since we will give a semantics-preserving translation from UF to DS in Section 3.2.

▶ **Definition 12.** Let $c : \mathsf{Uni}\ D$ and $\alpha : \mathsf{Info}\ c \to E$. The uniform code $(c, \alpha) : \mathsf{UF}\ D\ E$ induces a functor $[\![\ c, \alpha\ ]\!] : \mathsf{Fam}\ D \to \mathsf{Fam}\ E$ by $[\![\ c, \alpha\ ]\!]\ Z = \mathsf{Fam}(\alpha)\ ([\![\ c\ ]\!]_{\mathsf{Uni}}\ Z, [\![\ c\ ]\!]_{\mathsf{Info}}\ Z)$ where $[\![\ \_\ ]\!]_{\mathsf{Uni}} : \mathsf{Uni}\ D \to \mathsf{Fam}\ D \to \mathsf{Set}$ and $[\![\ \_\ ]\!]_{\mathsf{Info}} : \big(c : \mathsf{Uni}\ D\big) \to \big(Z : \mathsf{Fam}\ D\big) \to [\![\ c\ ]\!]_{\mathsf{Uni}}\ Z \to \mathsf{Info}\ c$ are simultaneously defined by induction on c:

$$\begin{aligned}
&[\![\ \iota_{\mathsf{UF}}\ ]\!]_{\mathsf{Uni}}\ (U,T) = 1 && [\![\ \iota_{\mathsf{UF}}\ ]\!]_{\mathsf{Info}}\ (U,T)\ \star = \star\\
&[\![\ \sigma_{\mathsf{UF}}\ c\ A\ ]\!]_{\mathsf{Uni}}\ (U,T) = && [\![\ \sigma_{\mathsf{UF}}\ c\ A\ ]\!]_{\mathsf{Info}}\ (U,T)\ (x,a) =\\
&\quad \big(\Sigma x : [\![\ c\ ]\!]_{\mathsf{Uni}}\ (U,T)\big)(A([\![\ c\ ]\!]_{\mathsf{Info}}\ (U,T)\ x)) && \quad ([\![\ c\ ]\!]_{\mathsf{Info}}\ (U,T)\ x, a)\\
&[\![\ \ \delta_{\mathsf{UF}}\ c\ A\ \ ]\!]_{\mathsf{Uni}}\ (U,T) = && [\![\ \delta_{\mathsf{UF}}\ c\ A\ ]\!]_{\mathsf{Info}}\ (U,T)\ (x,g) =\\
&\quad \big(\Sigma x : [\![\ c\ ]\!]_{\mathsf{Uni}}\ (U,T)\big)(A([\![\ c\ ]\!]_{\mathsf{Info}}\ (U,T)\ x) \to U) && \quad ([\![\ c\ ]\!]_{\mathsf{Info}}\ (U,T)\ x, T \circ g)
\end{aligned}$$

▶ **Example 13.** Decoding $c_{\mathsf{W}\ S\ P,\mathsf{UF}}$ from Example 10, we see that

$$[\![\ c_{\mathsf{W}\ S\ P,\mathsf{UF}}\ ]\!]_{\mathsf{Uni}}\ (U,T) = \big(\Sigma(\star, s) : 1 \times S\big)(P(s) \to U)$$

which is isomorphic to the domain of the W-type constructor sup, but this time nested the other way compared to the decoding of $c_{\mathsf{W}\ S\ P}$ in Example 5. By Theorem 18, we will have $[\![\ c +_{\mathsf{UF}} d\ ]\!]\ Z \cong [\![\ c\ ]\!]\ Z\ + [\![\ d\ ]\!]\ Z$, where the right hand side uses the coproduct of families. Hence $c_{2,\mathsf{UF}} +_{\mathsf{UF}} c_{\mathsf{W},\mathsf{UF}}$ from Example 11 decodes correctly.

3.2 Embedding of UF into DS

We embed UF into DS, i.e. we give a translation of codes which is *semantics-preserving* in that the decoding of a code is isomorphic to the decoding of its translation. Since UF codes are "backwards" compared to DS codes, this embedding resembles the well-known accumulator based algorithm for reversing a list. Define $\mathsf{accUFtoDS} : \big(c : \mathsf{Uni}\ D\big) \to (\mathsf{Info}\ c \to \mathsf{DS}\ D\ E) \to \mathsf{DS}\ D\ E$ (the second argument is the accumulator) by

$$\begin{aligned}
\mathsf{accUFtoDS}\ \iota_{\mathsf{UF}}\ F &= F\ \star\\
\mathsf{accUFtoDS}\ (\sigma_{\mathsf{UF}}\ c\ A)\ F &= \mathsf{accUFtoDS}\ c\ (\gamma \mapsto \sigma\ (A\ \gamma)\ (a \mapsto F\ (\gamma, a)))\\
\mathsf{accUFtoDS}\ (\delta_{\mathsf{UF}}\ c\ A)\ F &= \mathsf{accUFtoDS}\ c\ (\gamma \mapsto \delta\ (A\ \gamma)\ (h \mapsto F\ (\gamma, h)))
\end{aligned}$$

and define $\mathsf{UFtoDS} : \mathsf{UF}\ D\ E \to \mathsf{DS}\ D\ E$ by kicking things off with a ι:

$$\mathsf{UFtoDS}\ (c, \alpha) = \mathsf{accUFtoDS}\ c\ (\iota \circ \alpha)\ .$$

▶ **Proposition 14.** *The translation* UFtoDS *is an embedding, i.e. for every* $c : \mathsf{UF}\ D\ E$ *and* $Z : \mathsf{Fam}\ D$*, we have* $[\![\ \mathsf{UFtoDS}\ c\ ]\!]\ Z \cong [\![\ c\ ]\!]\ Z$*.* ◀

3.3 Coproducts of Uniform Codes

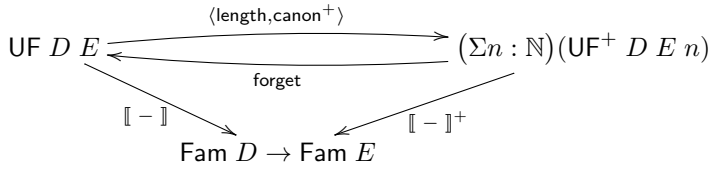
The coproduct $c +_{\mathsf{DS}} d \coloneqq \sigma\ 2\ (\mathsf{tt} \mapsto c; \mathsf{ff} \mapsto d)$ of two DS codes is not in general the embedding of a uniform code, even if c and d are, as c and d may still have different shapes. Hence we cannot immediately use the same construction to define coproducts of uniform codes, but we note that whenever c and d do have the same shape, this construction still works. Our plan for constructing coproducts of uniform codes is then to find equivalent replacements of the summands, such that the new pair has a common shape, and then using the standard coproduct. To this end, we introduce an $\mathbb{N}$-indexed variant $\mathsf{UF}^+\ D\ E\ n = \big(\Sigma c : \mathsf{Uni}^+\ D\ n\big)(\mathsf{Info}^+\ c \to E)$ of UF for this section only. There are two differences between UF^+ and UF: firstly, UF^+ is indexed by the length n of its codes, and secondly in UF^+ the δ_{UF} and σ_{UF} codes are replaced by a combined code

$$\begin{aligned}
\delta\sigma : \big(c : \mathsf{Uni}^+\ D\ n\big) \to \big(A : \mathsf{Info}^+\ c \to \mathsf{Set}\big) \to\\
\big((\gamma : \mathsf{Info}^+\ c) \to A\ \gamma \to \mathsf{Set}\big) \to \mathsf{Uni}^+\ D\ (\mathsf{suc}\ n)
\end{aligned}$$

with $\mathsf{Info}^+\ (\delta\sigma\ c\ A\ B) = \big(\Sigma\gamma : \mathsf{Info}^+\ c\big)\big(\Sigma x : A\ \gamma\big)(B\ \gamma\ x \to D)$. The code $\delta\sigma$ should be thought of as a δ_{UF} code followed by a σ_{UF} code. We can recover "ordinary" σ_{UF} and δ_{UF} by $\sigma_+\ c\ A \coloneqq \delta\sigma\ c\ A\ (\_,\_ \mapsto 0)$ and $\delta_+\ c\ B \coloneqq \delta\sigma\ c\ (\_ \mapsto 1)(\gamma,\_ \mapsto B\,\gamma)$. We have just informally described translations $\mathsf{forget} : \mathsf{UF}^+\ D\ E\ n \to \mathsf{UF}\ D\ E$ and $\mathsf{canon}^+ : \big(c : \mathsf{UF}\ D\ E\big) \to \mathsf{UF}^+\ D\ E\ (\mathsf{length}\,c)$, where length counts the depth of the code c. A decoding $[\![-]\!]^+$ can be defined for UF^+ along the lines for the one for UF (alternatively, Proposition 15(ii) below can be used as a definition).

▶ **Proposition 15.** *Let $D, E : \mathsf{Set}_1$ and $Z : \mathsf{Fam}\ D$. If $c : \mathsf{UF}\ D\ E$ and $d : \mathsf{UF}^+\ D\ E\ n$, then (i) $[\![\ \mathsf{canon}^+\ c\ ]\!]^+\ Z \cong [\![\ c\ ]\!]\ Z$; and (ii) $[\![\ \mathsf{forget}\ d\ ]\!]\ Z \cong [\![\ d\ ]\!]^+\ Z$.* ◀

This proposition can be summed up in the following commuting diagram:



Next, note $[\![\ \delta\sigma\ c\ 1\ (\_ \mapsto 0)\ ]\!]^+\ Z \cong [\![\ c\ ]\!]^+\ Z$. Thus we can pad out $c : \mathsf{UF}^+\ D\ E\ n$ to $\mathsf{pad}_k\ c : \mathsf{UF}^+\ D\ E\ (n + k + 1)$ without changing the meaning of the code:

▶ **Lemma 16.** *Let $k : \mathbb{N}$. There is an operation $\mathsf{pad}_k : \mathsf{UF}^+\ D\ E\ n \to \mathsf{UF}^+\ D\ E\ (n + k + 1)$ such that $[\![\ \mathsf{pad}_k\ c\ ]\!]^+\ Z \cong [\![\ c\ ]\!]^+\ Z$ for every $Z : \mathsf{Fam}\ D$.* ◀

Since all UF^+ codes of the same length also are of the same shape, it is now easy to form coproducts of such codes. Define $\_ +_+ \_ : \mathsf{UF}^+\ D\ E\ n \to \mathsf{UF}^+\ D\ E\ n \to \mathsf{UF}^+\ D\ E\ (\mathsf{suc}\,n)$ by $(c, \alpha) +_+ (d, \beta) = (c +_{\mathsf{Uni}} d, [\alpha, \beta] \circ (c +_{\mathsf{Info}} d))$ where $\_ +_{\mathsf{Uni}} \_$ is defined by

$$\iota_+ +_{\mathsf{Uni}} \iota_+ = \sigma_+\ \iota_+\ (\_ \mapsto 2)$$
$$(\delta\sigma\ c\ A\ B) +_{\mathsf{Uni}} (\delta\sigma\ d\ A'\ B') = \delta\sigma\ (c +_{\mathsf{Uni}} d)\ ([A, A'] \circ (c +_{\mathsf{Info}} d))\ ([B, B'] \circ (c +_{\mathsf{Info}} d))$$

simultaneously with a map $(c +_{\mathsf{Info}} d) : \mathsf{Info}^+\ (c +_{\mathsf{Uni}} d) \to \mathsf{Info}^+\ c + \mathsf{Info}^+\ d$, whose definition is similar. Note that we did not need to consider the definition of e.g. $\iota_+ +_{\mathsf{Uni}} (\delta\sigma\ c\ A\ B)$ as these summands cannot possibly have the same length.

▶ **Lemma 17.** *For all $c, d : \mathsf{UF}^+\ D\ E\ n$ and $Z : \mathsf{Fam}\ D$ we have $[\![\ c +_+ d\ ]\!]^+\ Z \cong [\![\ c\ ]\!]^+\ Z + [\![\ d\ ]\!]^+\ Z$, where the right hand side is a coproduct of families.* ◀

Putting everything together, we have:

▶ **Theorem 18.** *Let $D, E : \mathsf{Set}_1$. Define $\_ +_{\mathsf{UF}} \_ : \mathsf{UF}\ D\ E \to \mathsf{UF}\ D\ E \to \mathsf{UF}\ D\ E$ by $c +_{\mathsf{UF}} d = \mathsf{forget}\ (\mathsf{canon}^+\ c +_+ \mathsf{canon}^+\ d)$. Then $[\![\ c +_{\mathsf{UF}} d\ ]\!]\ Z \cong [\![\ c\ ]\!]\ Z + [\![\ d\ ]\!]\ Z$.* ◀

3.4 Composition of uniform Codes

Recall Section 2.2, where composition of DS codes followed from a power operation which —because of the dependency arising in its attempted construction— we could not define. Fortunately, in UF, a power operator is definable! Composition is here —as for DS— facilitated by a conjunction of the power operation and a bind operator. A full bind operation for UF is not definable since the grafting of uniform trees into a uniform tree may not be uniform since the trees may differ in height (i.e. UF is not a monad). However, for composition, it suffices to graft trees of the *same* height. Define $- \gg\!=[- \longrightarrow -] : \big(c : \mathsf{Uni}\ D\big) \to (\mathsf{Info}\ c \to$

$\mathsf{Set}) \to \mathsf{Uni}\ D \to \mathsf{Uni}\ D$, together with $(c \gg\!=\![E \longrightarrow d])_{\mathsf{Info}} : \mathsf{Info}\ (c \gg\!=\![E \longrightarrow d]) \to \big(\Sigma x : \mathsf{Info}\ c\big)(E\, x \to \mathsf{Info}\ d)$, which explains the meaning of $c \gg\!=\![E \longrightarrow d]$ at the level of Info. We write $\gg\!=_{\mathsf{Info},0}$ and $\gg\!=_{\mathsf{Info},1}$ for the first and second projection of $(c \gg\!=\![E \longrightarrow d])_{\mathsf{Info}}$ respectively, inferring the other arguments from context:

$$
\begin{aligned}
&c \gg\!=\![E \longrightarrow \iota_{\mathsf{UF}}] = c\\
&c \gg\!=\![E \longrightarrow \sigma_{\mathsf{UF}}\ d\ A]\\
&\qquad = \sigma_{\mathsf{UF}}\ (c \gg\!=\![E \longrightarrow d])(\gamma \mapsto \big(e : E(\gg\!=_{\mathsf{Info},0}\ \gamma)\big) \to A(\gg\!=_{\mathsf{Info},1}\ \gamma\, e))\\
&c \gg\!=\![E \longrightarrow \delta_{\mathsf{UF}}\ d\ A] =\\
&\qquad = \delta_{\mathsf{UF}}\ (c \gg\!=\![E \longrightarrow d])(\gamma \mapsto \big(\Sigma e : E(\gg\!=_{\mathsf{Info},0}\ \gamma)\big)A(\gg\!=_{\mathsf{Info},1}\ \gamma\, e))
\end{aligned}
$$

$$
\begin{aligned}
&(c \gg\!=\![E \longrightarrow \iota_{\mathsf{UF}}])_{\mathsf{Info}}\ x = (x, (\_ \mapsto \star))\\
&(c \gg\!=\![E \longrightarrow \sigma_{\mathsf{UF}}\ d\ A])_{\mathsf{Info}}\ (x, g) = (\gg\!=_{\mathsf{Info},0}\ \ x, e \mapsto (\gg\!=_{\mathsf{Info},0}\ \ x\ e, g\ e))\\
&(c \gg\!=\![E \longrightarrow \delta_{\mathsf{UF}}\ d\ A])_{\mathsf{Info}}\ (x, g) = (\gg\!=_{\mathsf{Info},0}\ \ x, e \mapsto (\gg\!=_{\mathsf{Info},0}\ \ x\ e, (a \mapsto g\ (e, a)))
\end{aligned}
$$

This definition is validated by the following proposition:

▶ **Proposition 19.** *There is an equivalence*

$$[\![\ c \gg\!=\![E \longrightarrow d], (d \gg\!=\![E \longrightarrow d])_{\mathsf{Info}}\ ]\!] \cong ([\![\ c, \mathsf{id}\ ]\!]) \gg\!=_{\mathsf{Fam}} (e \mapsto ((E\ e) \longrightarrow_{\mathsf{Fam}} [\![\ d, \mathsf{id}\ ]\!])) \quad \blacktriangleleft$$

▶ Remark. While is not possible to derive a bind operator from $\_ \gg\!=\![\_ \longrightarrow \_]$, we do obtain a power operator with the right universal property by

$$A \longrightarrow (c, f) \coloneqq (\iota_{\mathsf{UF}} \gg\!=\![(\_ \mapsto A) \longrightarrow c], (\gamma \mapsto f \circ \gg\!=_{\mathsf{Info},1}))\ .$$

(This fact will not be needed in the proof of composition in Theorem 20.)

We can now define composition for UF codes in a fashion similar to Theorem 8, except that we separate the action of the first component of a code and take care of the second component in a second step:

▶ **Theorem 20.** *The operations*

$$
\begin{aligned}
&\_ \bullet_{\mathsf{Uni}} \_ : \mathsf{Uni}\ D \to \mathsf{UF}\ C\ D \to \mathsf{Uni}\ C\\
&(\_ \bullet_{\mathsf{Info}} \_) : \big(c : \mathsf{Uni}\ D\big) \to \big(R : \mathsf{UF}\ C\ D\big) \to \mathsf{Info}\ (c \bullet_{\mathsf{Uni}} R) \to \mathsf{Info}\ c
\end{aligned}
$$

simultaneously defined by

$$
\begin{aligned}
\iota_{\mathsf{UF}} \bullet_{\mathsf{Uni}} R &= \iota_{\mathsf{UF}}\\
(\sigma_{\mathsf{UF}}\ c\ A) \bullet_{\mathsf{Uni}} R &= \sigma_{\mathsf{UF}}\ (c \bullet_{\mathsf{Uni}} R)\ (A \circ (c \bullet_{\mathsf{Info}} R))\\
(\delta_{\mathsf{UF}}\ c\ A) \bullet_{\mathsf{Uni}} (d, \beta) &= (c \bullet_{\mathsf{Uni}} (d, \beta)) \gg\!=\![(A \circ (c \bullet_{\mathsf{Info}} (d, \beta))) \longrightarrow d]\\
(\iota_{\mathsf{UF}} \bullet_{\mathsf{Info}} R)\ x &= x\\
((\sigma_{\mathsf{UF}}\ c\ A) \bullet_{\mathsf{Info}} R)\ (x, y) &= ((c \bullet_{\mathsf{Info}} R)\ x, y)\\
((\delta_{\mathsf{UF}}\ c\ A) \bullet_{\mathsf{Info}} (d, \beta))\ x &= ((c \bullet_{\mathsf{Info}} (d, \beta))\ (\gg\!=_{\mathsf{Info},0}\ \ x, \beta \circ (\gg\!=_{\mathsf{Info},1}\ \ x)))
\end{aligned}
$$

make $\_ \bullet \_ : \mathsf{UF}\ D\ E \to \mathsf{UF}\ C\ D \to \mathsf{UF}\ C\ E$ *a composition operation for* UF *codes, where*

$$(c, \alpha) \bullet (d, \beta) = (c \bullet_{\mathsf{Uni}} (d, \beta), \alpha \circ (c \bullet_{\mathsf{Info}} (d, \beta)))\ .$$

◀

▶ **Example 21.** If we compose c_{2W} from Example 11 with the "upgraded" code $c_{\mathsf{W}\,\mathbb{N}\,\mathsf{Fin}}$ from Example 10, we get a code for a universe where each constructor now takes a list of inductive arguments, with decoding the product of the decodings. Up to an isomorphism relating coproducts of compositions with compositions of coproducts, the resulting code is $c_{\mathsf{2W}} \bullet c_{\mathsf{W}\,\mathbb{N}\,\mathsf{Fin}} \cong c_{2,\mathsf{UF}} +_{\mathsf{UF}} c'_{W,\mathsf{UF}}$, where $c_{2,\mathsf{UF}}$ is as before, and

$$\begin{aligned} c'_{W,\mathsf{UF}} = (\delta_{\mathsf{UF}}\ (\sigma_{\mathsf{UF}}\ c_{\mathsf{W}\,\mathbb{N}\,\mathsf{Fin}}\ ((\star, n, Y) &\mapsto ((x : \mathsf{Fin}\,n) \to Y\,x) \to \mathbb{N})) \\ ((\star, n, Y, e) &\mapsto (\Sigma y : (x : \mathsf{Fin}\,n) \to Y\,x)\mathsf{Fin}\,(e\,y)), \\ ((\star, n, Y, e, B) &\mapsto (\Sigma y : (x : \mathsf{Fin}\,n) \to Y\,x)(w : \mathsf{Fin}\,(e\,y)) \to B\,(y, w))) \ . \end{aligned}$$

4 Polynomial Codes PN for Inductive-Recursive Definitions

We saw in Section 2.2 that composition for Dybjer-Setzer codes requires a power operator. However, simply adding a code for powers means that $\mathsf{DS}\ D\ \_$ is no longer a monad, and the bind operation was crucial for constructing composition. Hence, further adjustments are required. Following this line of thought results in a system including sums and type-indexed products. For this reason, we call it *polynomial inductive-recursive definitions*, and denote it by PN. It was originally invented by the second author in order to make induction-recursion resemble the descriptions of datatypes in Chapman et al. [3]. Just like uniform codes, polynomial codes are presented as a two-level definition which itself is an inductive-recursive definition:

▶ **Definition 22.** Let $D, E : \mathsf{Set}_1$. The large type $\mathsf{PN}\ D\ E : \mathsf{Set}_1$ of *polynomial codes for induction-recursion* is defined by $\mathsf{PN}\ D\ E \coloneqq (\Sigma c : \mathsf{Poly}\,\mathsf{D})(\mathsf{Info}\,\mathsf{c} \to E)$, where $\mathsf{Poly}\,\mathsf{D} : \mathsf{Set}_1$ and $\mathsf{Info} : \mathsf{Poly}\,\mathsf{D} \to \mathsf{Set}_1$ are mutually defined by

$$\begin{array}{rll} \mathsf{id}_{\mathsf{PN}} : \mathsf{Poly}\,\mathsf{D} & & \mathsf{Info}\,\mathsf{id}_{\mathsf{PN}} = D \\ \mathsf{con} : (A : \mathsf{Set}) \to \mathsf{Poly}\,\mathsf{D} & & \mathsf{Info}\,(\mathsf{con}\,\mathsf{A}) = A \\ \mathsf{sig} : (S : \mathsf{Poly}\,\mathsf{D}) \to (\mathsf{Info}\,\mathsf{S} \to \mathsf{Poly}\,\mathsf{D}) \to \mathsf{Poly}\,\mathsf{D} & & \mathsf{Info}\,(\mathsf{sig}\,\mathsf{S}\,\mathsf{F}) = (\Sigma x : \mathsf{Info}\,\mathsf{S})(\mathsf{Info}\,(\mathsf{F}\,\mathsf{x})) \\ \mathsf{pi} : (A : \mathsf{Set}) \to (A \to \mathsf{Poly}\,\mathsf{D}) \to \mathsf{Poly}\,\mathsf{D} & & \mathsf{Info}\,(\mathsf{pi}\,\mathsf{A}\,\mathsf{F}) = (x : A) \to \mathsf{Info}\,(\mathsf{F}\,\mathsf{x}) \end{array}$$

Warning: polynomial codes should not be confused with polynomial functors [9, 10]! We use the same name Info as in uniform codes for the function computing the information represented by a code. The code $\mathsf{id}_{\mathsf{PN}}$ represents the identity functor, $\mathsf{con}\ A$ the functor constantly returning index type A, $\mathsf{sig}\ S\ F$ represents a dependent coproduct of functors, and $\mathsf{pi}\ A\ F$ represents an A-indexed dependent product of functors. Observe that $\mathsf{PN}\,D\,\_$ is again, like $\mathsf{UF}\,D\,\_$, functorial by function composition.

▶ **Example 23** (W-types, again)**.** We revisit Examples 2 and 10. For $S : \mathsf{Set}$, $P : S \to \mathsf{Set}$ the polynomial code for the W-type $\mathsf{W}\ S\ P$ is $(c_{\mathsf{W}\ S\ P,\mathsf{PN}}, \_ \mapsto \star) : \mathsf{PN}\ 1\ 1$ where $c_{\mathsf{W}\ S\ P,\mathsf{PN}} = \mathsf{sig}\ (\mathsf{con}\ S)(s \mapsto\ \mathsf{pi}\ (P\ s)\ (\_ \mapsto \mathsf{id}_{\mathsf{PN}}))$. Again this can be upgraded to a PN Set Set code, applying $T : U \to \mathsf{Set}$ everywhere in the structure, by replacing the trivial map $(\_ \mapsto \star) : \mathsf{Info}\,\mathsf{c}_{\mathsf{W}\ \mathsf{S}\ \mathsf{P},\mathsf{PN}} \to 1$ by the map $((s, Y) \mapsto (c : P(s)) \to Y\,x) : \mathsf{Info}\,\mathsf{c}_{\mathsf{W}\ \mathsf{S}\ \mathsf{P},\mathsf{PN}} \to \mathsf{Set}$.

▶ **Example 24** (A universe closed under W-types, again)**.** We also revisit Example 3 again. A polynomial code $(c_{\mathsf{2W},\mathsf{PN}}, \alpha) : \mathsf{PN}\ \mathsf{Set}\ \mathsf{Set}$ for a universe containing 2, closed under W-types is given by $c_{\mathsf{2W},\mathsf{PN}} : \mathsf{Poly}\ \mathsf{Set}$ and $\alpha_{\mathsf{2W},\mathsf{PN}} : \mathsf{Info}\,\mathsf{c}_{\mathsf{2W},\mathsf{PN}} \to \mathsf{Set}$ where

$$c_{\mathsf{2W},\mathsf{PN}} = \mathsf{sig}\ (\mathsf{con}\ \{\mathsf{two}, \mathsf{w}\})(\mathsf{two} \mapsto \mathsf{con}\ 1; \mathsf{w} \mapsto \mathsf{sig}\ \mathsf{id}_{\mathsf{PN}}\ (X \mapsto \mathsf{pi}\ X\ (\_ \mapsto \mathsf{id}_{\mathsf{PN}})))$$

and $\alpha_{\mathsf{2W},\mathsf{PN}}$ is defined by $\alpha_{\mathsf{2W},\mathsf{PN}}(\mathsf{two}, x) = 2$ and $\alpha_{\mathsf{2W},\mathsf{PN}}(\mathsf{w}, (A, B)) = \mathsf{W}\ S\ P$.

▶ Remark. One obtains a weaker system by replacing the pi code by a code $\mathsf{pow} : \mathsf{Set} \to \mathsf{Poly}\ D \to \mathsf{Poly}\ D$ with $\mathsf{Info}\,(\mathsf{pow}\,A\,c) = A \to \mathsf{Info}\,c$. In the full system, such a code can be defined by $\mathsf{pow}\ A\ c \coloneqq \mathsf{pi}\ A\ (\_ \mapsto c)$. The weaker system also enjoys composition, and the embedding of Dybjer-Setzer codes in Section 4.1 factors through the system with powers only. Semantically, the stronger system is just as easy to handle (see Theorem 27 below).

Polynomial codes in $\mathsf{PN}\ D\ E$ decode to functors $\mathsf{Fam}\ D \to \mathsf{Fam}\ E$ in the following way:

▶ **Definition 25.** Let $c : \mathsf{Poly}\ D$ and $\alpha : \mathsf{Info}\,c \to E$. The polynomial code $(c, \alpha) : \mathsf{PN}\ D\ E$ induces a functor $[\![\ c, \alpha\ ]\!] : \mathsf{Fam}\ D \to \mathsf{Fam}\ E$ by $[\![\ c, \alpha\ ]\!]\ Z = \mathsf{Fam}(\alpha)\ ([\![\ c\ ]\!]_0\ Z, [\![\ c\ ]\!]_{\mathsf{info}}\ Z)$ where $[\![\ c\ ]\!]_0 : \mathsf{Fam}\ D \to \mathsf{Set}$ and $[\![\ c\ ]\!]_{\mathsf{info}} : (X : \mathsf{Fam}\ D) \to [\![\ c\ ]\!]_0\ X \to \mathsf{Info}\,c$ are simultaneously defined by induction on c:

$$[\![\ \mathsf{id_{PN}}\ ]\!]_0\ (U,T) = U \qquad [\![\ \mathsf{id_{PN}}\ ]\!]_{\mathsf{info}}\ (U,T)\ x = T\,x$$
$$[\![\ \mathsf{con}\ A\ ]\!]_0\ X = A \qquad [\![\ \mathsf{con}\ A\ ]\!]_{\mathsf{info}}\ X\ a = a$$
$$[\![\ \mathsf{sig}\ S\ F\ ]\!]_0\ (U,T) = \big(\Sigma s : [\![\ S\ ]\!]_0\ (U,T)\big)([\![\ F([\![\ S\ ]\!]_{\mathsf{info}}\ (U,T)\ s)\ ]\!]_0\ (U,T))$$
$$[\![\ \mathsf{sig}\ S\ F\ ]\!]_{\mathsf{info}}\ (U,T)\ (s,x) = ([\![\ S\ ]\!]_{\mathsf{info}}\ (U,T)\ s, [\![\ F\ ([\![\ S\ ]\!]_{\mathsf{info}}\ (U,T)\ s)\ ]\!]_{\mathsf{info}}\ (U,T)\ x)$$
$$[\![\ \mathsf{pi}\ A\ F\ ]\!]_0\ X = \big(x : A\big) \to [\![\ Fx\ ]\!]_0\ X \qquad [\![\ \mathsf{pi}\ A\ F\ ]\!]_{\mathsf{info}}\ X\ g = (a \mapsto [\![\ (Fa)\ ]\!]_{\mathsf{info}}\ X\ (g\ a))$$

▶ **Example 26.** Decoding $c_{\mathsf{W}\ S\ P,\mathsf{PN}}$ from Example 23, we get

$$[\![\ c_{\mathsf{W}\ S\ P,\mathsf{PN}}\ ]\!]_0\ (U,T) = \big(\Sigma s : S\big)(P(s) \to U)$$

this time matching the domain of the W-type constructor sup strictly. Similarly decoding $(c_{\mathsf{2W},\mathsf{PN}}, \alpha_{\mathsf{2W},\mathsf{PN}})$ from Example 24 we again get the same result as in Example 6.

Since we did not exhibit PN as a subsystem of DS, we cannot rely on Dybjer and Setzer's proof of soundness, i.e. that initial algebras of the corresponding functors exist in their model. We can, however, extend their proof to polynomial codes[2]:

▶ **Theorem 27.** *Working in ZFC, assume the existence of a Mahlo cardinal* M *and a 1-inaccessible cardinal* I *above it. Then there is a set-theoretic model of Martin-Löf Type Theory +* PN *where types* $A : \mathsf{Set}$ *are interpreted as sets in* V_{M} *and large types* $D : \mathsf{Set}_1$ *are interpreted as sets in* V_{I} *(here* V_α *is the cumulative hierarchy). In this model, all functors* $[\![\ c\ ]\!] : \mathsf{Fam}\ D \to \mathsf{Fam}\ D$ *arising from polynomial codes* $c : \mathsf{PN}\ D\ D$ *have initial algebras.* ◀

Note that the existence of large cardinals is only needed for the soundness proof, and not for working within the theory itself. The same situation applies to Dybjer and Setzer's DS.

4.1 Embedding of DS into PN

▶ **Proposition 28.** *The map* $\mathsf{DStoPN} : \mathsf{DS}\ D\ E \to \mathsf{PN}\ D\ E$ *given by* $\mathsf{DStoPN}\,c = (\mathsf{toP}\,c, \mathsf{tol}\,c)$ *where* $\mathsf{toP} : \mathsf{DS}\ D\ E \to \mathsf{Poly\,D}$ *and* $\mathsf{tol} : \big(c : \mathsf{DS}\ D\ E\big) \to \mathsf{Info}\,(\mathsf{toP}\,c) \to E$ *are defined by*

$$\mathsf{toP}(\iota\ e) = \mathsf{con}\ 1 \qquad \mathsf{tol}(\iota\ e)\ \star = e$$
$$\mathsf{toP}(\sigma\ A\ f) = \mathsf{sig}\ (\mathsf{con}A)\ (\mathsf{toP} \circ f) \qquad \mathsf{tol}(\sigma\ A\ f)\ (a,x) = \mathsf{tol}\ (f\ a)\ x$$
$$\mathsf{toP}(\delta\ A\ F) = \mathsf{sig}\ (\mathsf{pi}\ A\ (\_ \mapsto \mathsf{id_{PN}}))\ (\mathsf{toP} \circ F) \qquad \mathsf{tol}(\delta\ A\ F)\ (g,x) = \mathsf{tol}\ (F\ g)\ x$$

is semantics-preserving. ◀

We conjecture that this embedding is strict, i.e. that there is a code $c : \mathsf{PN}\ D\ E$ with $[\![\ c\ ]\!] \not\simeq [\![\ \mathsf{DStoPN}\ d\ ]\!]$ for every $d : \mathsf{DS}\ D\ E$ for some $D, E : \mathsf{Set}_1$.

[2] We require a little bit more from the metatheory: Dybjer and Setzer [8] require I to be 0-inaccessible only. But existence of I is a mild assumption compared to the existence of M.

4.2 Composition of Polynomial Codes

Composition for PN codes can be defined following the same pattern as in Proposition 8, where we constructed composition for DS codes using the assumption of a power operation, and the fact that DS is a monad. The system PN has a power operation using the pi constructor, and is a monad thanks to the sig constructor:

▶ **Proposition 29.** *For each* $D : \mathsf{Set}_1$, $\mathsf{PN}\ D$ *is a monad, i.e. there are terms* $\eta_{\mathsf{PN}} : E \to \mathsf{PN}\ D\ E$ *and* $\mu_{\mathsf{PN}} : \mathsf{PN}\ D\ (\mathsf{PN}\ D\ E) \to \mathsf{PN}\ D\ E$ *satisfying the monad laws. Furthermore, let* $(U,T) : \mathsf{Fam}\ D$. *Then* $[\![\ \eta_{\mathsf{PN}}(e)\ ]\!](U,T) = \eta_{\mathsf{Fam}}(e)$ *for every* $e : E$ *and , and* $[\![\ \mu_{\mathsf{PN}}(c)\ ]\!](U,T) = \mu_{\mathsf{Fam}}(\mathsf{Fam}([\![\ -\ ]\!](U,T))([\![\ c\ ]\!](U,T)))$ *for every* $c : \mathsf{PN}\ D\ (\mathsf{PN}\ D\ E)$.

Proof. We define $\eta_{\mathsf{PN}}(e) = (\mathsf{con}\ 1, \_ \mapsto e)$ and $\mu_{\mathsf{PN}}(c,\alpha) = (\mathsf{sig}\ c\ (\mathsf{fst} \circ \alpha), (x,y) \mapsto \mathsf{snd}\ (\alpha\ x)\ y)$. The equations in terms of the monad structure on Fam holds on the nose. ◀

Using the monad structure, we can define a "dependent bind" operation

$$\begin{aligned} \_ \gg\!=_{\mathsf{PN}} \_ &: \mathsf{PN}\ C\ D \to ((x : D) \to \mathsf{PN}\ C\ (E\,x)) \to \mathsf{PN}\ C\ ((\Sigma x : D)(E\,x)) \\ c \gg\!=_{\mathsf{PN}} h &= \mu_{\mathsf{PN}}(\mathsf{PN}(x \mapsto \mathsf{PN}(y \mapsto (x,y)))\ (h\,x)\ c) \end{aligned}$$

We also note that the pi constructor can be packaged up into the following "dependent power" operation for $S : \mathsf{Set}$ and $E : A \to \mathsf{Set}_1$:

$$\begin{aligned} \pi_{\mathsf{PN}} A &: (a : A) \to \mathsf{PN}\ D\ (E\,a) \to \mathsf{PN}\ D\ ((a : A) \to (E\,a)) \\ \pi_{\mathsf{PN}}\ A\ f &= (\mathsf{pi}\ A\ (\mathsf{fst} \circ f), (g \mapsto (a \mapsto \mathsf{snd}\,(f a)\ (g a)))) \end{aligned}$$

Using these ingredients, we can now define composition of PN codes:

▶ **Theorem 30.** *For* $c : \mathsf{Poly}\ D$ *and* $\alpha : \mathsf{Info}\,c \to E$ *and* $R : \mathsf{PN}\ C\ D$, *define* $(c,\alpha) \bullet R = \mathsf{PN}(\alpha)\,(c/R) : \mathsf{PN}\ C\ E$, *where* $\_/\_ : (c : \mathsf{Poly}\ E) \to \mathsf{PN}\ D\ E \to \mathsf{PN}\ D(\mathsf{Info}\,c)$ *is defined by*

$$\begin{aligned} \mathsf{id}_{\mathsf{PN}}/R &= R & (\mathsf{sig}\ c\ f)/R &= (c/R) \gg\!=_{\mathsf{PN}} (p \mapsto (f\,p)/R) \\ (\mathsf{con}\ A)/R &= (\mathsf{con} A, \mathsf{id}) & (\mathsf{pi}\ A\ f)/R &= \pi_{\mathsf{PN}}\ A\ (a \mapsto (f a)/R) \end{aligned}$$

Then $[\![\ R \bullet Q\ ]\!]\,(U,T) \cong [\![\ R\ ]\!]\,([\![\ Q\ ]\!]\,(U,T))$. ◀

▶ **Example 31.** Let us compose $c_{\mathsf{2W,PN}}$ from Example 24 with the "upgraded" code $c_{\mathsf{W}\ \mathbb{N}\ \mathsf{Fin,PN}}$ from Example 23. This time we get the code $\mathsf{sig}\ (\mathsf{con}\ \{\mathsf{two}, \mathsf{w}\})\ f$, where $f\ \mathsf{two} = \mathsf{con}\ 1$ and $f\ \mathsf{w} = \mathsf{sig}\ c_{\mathsf{W}\ \mathbb{N}\ \mathsf{Fin,PN}}\ ((n,Y) \mapsto \mathsf{pi}\ ((x : \mathsf{Fin}\,n) \to (Y\,x))\ (\_ \mapsto c_{\mathsf{W}\ \mathbb{N}\ \mathsf{Fin,PN}}))$.

5 Conclusions

Inductive-recursive definitions arise as initial algebras of endofunctors on Fam D, but the question of exactly which functors does not have a canonical answer. Dybjer and Setzer [7] gave one axiomatisation DS, which was adequate in the sense that it covered all examples "in the wild", and all functors represented in it could be shown to have initial algebras (in a sufficiently strong metatheory). We have presented two alternative axiomatisations UF and PN that retain these properties, but in addition are closed under composition. This opens up the field to find the *optimal* axiomatisation of inductive-recursive definitions. As a start, we hope to show in future work that both inclusions $\mathsf{UF} \hookrightarrow \mathsf{DS} \hookrightarrow \mathsf{PN}$ are strict.

Acknowledgements. We would like to thank Peter Hancock and Anton Setzer for inspiration and interesting discussions, and the reviewers for their suggestions and comments.

References

1 Michael Abbott, Thorsten Altenkirch, and Neil Ghani. Containers: Constructing strictly positive types. *TCS*, 342(1):3 – 27, 2005.

2 Thorsten Altenkirch, Neil Ghani, Peter Hancock, Conor McBride, and Peter Morris. Indexed containers. *Journal Functional Programming*, 25, 2015.

3 James Chapman, Pierre-Évariste Dagand, Conor McBride, and Peter Morris. The gentle art of levitation. In *ICFP 2010*, pages 3–14, 2010.

4 Peter Dybjer. Inductive families. *Formal aspects of computing*, 6(4):440–465, 1994.

5 Peter Dybjer. A general formulation of simultaneous inductive-recursive definitions in type theory. *Journal of Symbolic Logic*, 65(2), 2000.

6 Peter Dybjer and Anton Setzer. A finite axiomatization of inductive-recursive definitions. In *TLCA*, pages 129–146. Springer Verlag, 1999.

7 Peter Dybjer and Anton Setzer. Induction–recursion and initial algebras. *Annals of Pure and Applied Logic*, 124(1-3):1–47, 2003.

8 Peter Dybjer and Anton Setzer. Indexed induction–recursion. *Journal of logic and algebraic programming*, 66(1):1–49, 2006.

9 Nicola Gambino and Martin Hyland. Wellfounded trees and dependent polynomial functors. In *Types for Proofs and Programs*, pages 210–225, 2004.

10 Nicola Gambino and Joachim Kock. Polynomial functors and polynomial monads. *Mathematical Proceedings of the Cambridge Philosophical Society*, 154:153–192, 2013.

11 Neil Ghani and Peter Hancock. Containers, monads and induction recursion. *Mathematical Structures in Computer Science*, 26(1):89–113, 2016.

12 Peter Hancock. Private communication.

13 Per Martin-Löf. An intuitionistic theory of types: predicative part. In H.E. Rose and J.C. Shepherdson, editors, *Logic Colloquium '73, Proceedings of the Logic Colloquium*, volume 80 of *Studies in Logic and the Foundations of Mathematics*, pages 73–118. North-Holland, 1975.

14 Per Martin-Löf. *Intuitionistic type theory*, volume 1 of *Studies in Proof Theory*. Bibliopolis, 1984.

15 Robert Pollack. Dependently typed records in type theory. *Formal Aspects of Computing*, 13(3):386–402, 2002.