Variations on inductive-recursive definitions

Fredrik Nordvall Forsberg

University of Strathclyde, Glasgow

Agda Implementors' Meeting, Gothenburg, 12 May 2017




Joint work with Neil Ghani, Conor McBride, Peter Hancock and Stephan Spahn

An inductive definition

```
data Rose (A : Set) : Set where
  leaf : Rose A
  node : A → List (Rose A) → Rose A
```

We can represent Rose A by a functor $F_{\text{Rose}} : \text{Set} \to \text{Set}$:

$$F_{\text{Rose}}(X) = 1 + A \times \text{List}\, X$$

An inductive definition

```
data Rose (A : Set) : Set where
  leaf : Rose A
  node : A → List (Rose A) → Rose A
```

We can represent Rose A by a functor $F_{\mathsf{Rose}} : \mathsf{Set} \to \mathsf{Set}$:

$$F_{\mathsf{Rose}}(X) = 1 + A \times \mathsf{List}\, X$$

Rose A is the initial algebra of F_{Rose}.

An inductive-recursive definition

A universe closed inder $\mathbb{N}$ and Σ.

```
data U : Set
T : U → Set

data U where
  nat : U
  sig   : (a : U) → (b : T a → U) → U

T nat = ℕ
T (sig a b) = Σ (T a) (T ∘ b)
```

An inductive-recursive definition

A universe closed inder $\mathbb{N}$ and Σ.

```
data U : Set
T : U → Set

data U where
  nat : U
  sig   : (a : U) → (b : T a → U) → U

T nat = ℕ
T (sig a b) = Σ (T a) (T ∘ b)
```

U and T defined simultaneously.

An inductive-recursive definition

A universe closed inder $\mathbb{N}$ and Σ.

```
data U : Set
T : U → Set

data U where
  nat : U
  sig   : (a : U) → (b : T a → U) → U

T nat = ℕ
T (sig a b) = Σ (T a) (T ∘ b)
```

U and *T* defined simultaneously.

Also (U, T) is the initial algebra of a functor.

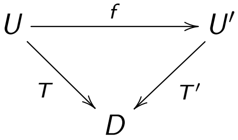
Category of families of Ds

The category Fam D for $D : \mathrm{Set}_1$:

- objects pairs (U, T) where

$$U : \mathrm{Set}$$
$$T : U \to D$$

- morphisms $(U, T) \to (U', T')$ are $f : U \to U'$ s.t.



commutes.

Note: Fam : Cat $\to$ Cat is a monad; D considered as discrete category.

An endofunctor on Fam Set

```
data U : Set where
  nat : U
  sig   : (a : U) → (b : T a → U) → U

T : U → Set
T nat = ℕ
T (sig a b) = Σ (T a) (T ∘ b)
```

is represented by F : Fam Set → Fam Set where

$$F(X, Q) = (1, \_ \mapsto \mathbb{N}) + \left(\left(\Sigma a : X\right)(Q\, a \to X), (a, b) \mapsto \Sigma\ (Q\, a)\ (Q \circ b)\right)$$

An endofunctor on Fam Set

```
data U : Set where
  nat : U
  sig   : (a : U) → (b : T a → U) → U

T : U → Set
T nat = ℕ
T (sig a b) = Σ (T a) (T ∘ b)
```

is represented by F : Fam Set → Fam Set where

$$F(X, Q) = (1, \_ \mapsto \mathbb{N}) + ((\Sigma a : X)(Q\,a \to X), (a, b) \mapsto \Sigma\ (Q\,a)\ (Q \circ b))$$

(U, T) is the initial algebra of F.

Representing inductive definitions

Not every functor defines a datatype. We want our inductive definitions to be strictly positive.

Representing inductive definitions

Not every functor defines a datatype. We want our inductive definitions to be strictly positive.

We can codify such definitions as follows (baby Dybjer-Setzer [1999, 2003, 2006]):

```
data ID : Set₁ where
  stop : ID
  side : (A : Set) → (c : A → ID) → ID
  ind  : (A : Set) → (c : ID) → ID
```

Representing inductive definitions

Not every functor defines a datatype. We want our inductive definitions to be strictly positive.

We can codify such definitions as follows (baby Dybjer-Setzer [1999, 2003, 2006]):

```
data ID : Set₁ where
  stop : ID
  side : (A : Set) → (c : A → ID) → ID
  ind  : (A : Set) → (c : ID) → ID
```

Each code gives rise to a functor:

$$
\begin{aligned}
&[\![-]\!] : \mathrm{ID} \to (\mathrm{Set} \to \mathrm{Set})\\
&[\![\mathrm{stop}]\!]\ X = 1\\
&[\![\mathrm{side}\ A\ c]\!]\ X = \big(\Sigma x : A\big)([\![c\ x]\!]\ X)\\
&[\![\mathrm{ind}\ A\ c]\!]\ X = (A \to X) \times [\![c]\!]\ X
\end{aligned}
$$

A code for List A

$$
\begin{array}{lll}
\mathsf{stop} & : & \mathsf{ID} \\
\mathsf{side} & : & (\mathtt{A} : \mathsf{Set}) \to \\
 & & (\mathtt{c} : \mathtt{A} \to \mathsf{ID}) \to \mathsf{ID} \\
\mathsf{ind} & : & (\mathtt{A} : \mathsf{Set}) \to \\
 & & (\mathtt{c} : \mathsf{ID}) \to \mathsf{ID}
\end{array}
$$

$$
\begin{aligned}
[\![\mathsf{stop}]\!]\ \mathtt{X} &= 1 \\
[\![\mathsf{side}\ A\ c]\!]\ \mathtt{X} &= \left(\Sigma x : A\right)[\![c\ x]\!]\ X \\
[\![\mathsf{ind}\ A\ c]\!]\ \mathtt{X} &= (\mathtt{A} \to \mathtt{X}) \times [\![c]\!]\ \mathtt{X}
\end{aligned}
$$

A code for List A

```
stop : ID
side : (A : Set) →
       (c : A → ID) → ID
ind  : (A : Set) →
       (c : ID) → ID
```

$$[\![\mathsf{stop}]\!]\ X = 1$$
$$[\![\mathsf{side}\ A\ c]\!]\ X = \left(\Sigma x : A\right)[\![c\ x]\!]\ X$$
$$[\![\mathsf{ind}\ A\ c]\!]\ X = (A \to X) \times [\![c]\!]\ X$$

The datatype

```
data List (A : Set) : Set where
  [] : List A
  _::_ : A → List A → List A
```

is represented by

$$c_{\mathsf{List}} = \mathsf{side}\,\{'[], '{::}\}\,('[] \mapsto\ \mathsf{stop};\ '{::} \mapsto \mathsf{side}\ A\ (\_ \mapsto \mathsf{ind}\ 1\ \mathsf{stop}))$$

A code for List A

```
stop : ID
side : (A : Set) →
       (c : A → ID) → ID
ind  : (A : Set) →
       (c : ID) → ID
```

$[\![\mathsf{stop}]\!]\ X = 1$

$[\![\mathsf{side}\ A\ c]\!]\ X = \left(\Sigma x : A\right)[\![c\ x]\!]\ X$

$[\![\mathsf{ind}\ A\ c]\!]\ X = (A \to X) \times [\![c]\!]\ X$

The datatype

```
data List (A : Set) : Set where
  [] : List A
  _::_ : A → List A → List A
```

is represented by

$$c_{\mathsf{List}} = \mathsf{side}\,\{\text{'}[], \text{'}{::}\}\,(\text{'}[] \mapsto \mathsf{stop};\ \text{'}{::} \mapsto \mathsf{side}\ A\ (\_ \mapsto \mathsf{ind}\ 1\ \mathsf{stop}))$$

Note: $\mathsf{side}\,\{\mathsf{tag}_c, \mathsf{tag}_d\}\,(\mathsf{tag}_c \mapsto c;\ \mathsf{tag}_d \mapsto d)$ for encoding coproducts of codes.

Representing inductive-recursive definitions

Dybjer-Setzer codes for functors Fam $D \to$ Fam E:

Representing inductive-recursive definitions

Dybjer-Setzer codes for functors Fam $D \to$ Fam E:

```
data ID : Set₁ where
  stop :  ID
  side : (A : Set) → (c : A → ID) → ID
  ind  : (A : Set) → (c : ID) → ID
```

Representing inductive-recursive definitions

Dybjer-Setzer codes for functors Fam $D \to$ Fam E:

```
data DS (D E : Set₁) : Set₁ where
  stop :  DS D E
  side : (A : Set) → (c : A → DS D E) → DS D E
  ind  : (A : Set) → (c : DS D E) → DS D E
```

Representing inductive-recursive definitions

Dybjer-Setzer codes for functors Fam $D \to$ Fam E:

```
data DS (D E : Set₁) : Set₁ where
  ι : E → DS D E
  side : (A : Set) → (c : A → DS D E) → DS D E
  ind : (A : Set) → (c : DS D E) → DS D E
```

Representing inductive-recursive definitions

Dybjer-Setzer codes for functors Fam $D \to$ Fam E:

```
data DS (D E : Set₁) : Set₁ where
  ι : E → DS D E
  σ : (A : Set) → (c : A → DS D E) → DS D E
  ind : (A : Set) → (c : DS D E) → DS D E
```

Representing inductive-recursive definitions

Dybjer-Setzer codes for functors Fam $D \to$ Fam E:

```
data DS (D E : Set₁) : Set₁ where
  ι : E → DS D E
  σ : (A : Set) → (c : A → DS D E) → DS D E
  δ : (A : Set) → (c : (A → D) → DS D E) → DS D E
```

Representing inductive-recursive definitions

Dybjer-Setzer codes for functors $\mathsf{Fam}\ D \to \mathsf{Fam}\ E$:

```
data DS (D E : Set₁) : Set₁ where
  ι : E → DS D E
  σ : (A : Set) → (c : A → DS D E) → DS D E
  δ : (A : Set) → (c : (A → D) → DS D E) → DS D E
```

$$[\![\_]\!] : \mathsf{DS}\ D\ E \to \mathsf{Fam}\ D \to \mathsf{Fam}\ E$$

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

Representing inductive-recursive definitions

Dybjer-Setzer codes for functors Fam $D \to$ Fam E:

```
data DS (D E : Set₁) : Set₁ where
  ι : E → DS D E
  σ : (A : Set) → (c : A → DS D E) → DS D E
  δ : (A : Set) → (c : (A → D) → DS D E) → DS D E
```

$$[\![\_]\!] : \mathsf{DS}\ D\ E \to \mathsf{Fam}\ D \to \mathsf{Fam}\ E$$

coproducts in Fam D

$$[\![\iota\ e]\!]\ (U, T) = (1, \star \mapsto e)$$
$$[\![\sigma\ A\ f]\!]\ (U, T) = \big(\Sigma a : A\big)([\![f\ a]\!]\ (U, T))$$
$$[\![\delta\ A\ F]\!]\ (U, T) = \big(\Sigma g : A \to U\big)([\![F\ (T \circ g)]\!]\ (U, T))$$

Representing inductive-recursive definitions

Dybjer-Setzer codes for functors Fam $D \to$ Fam E:

```
data DS (D E : Set₁) : Set₁ where
  ι : E → DS D E
  σ : (A : Set) → (c : A → DS D E) → DS D E
  δ : (A : Set) → (c : (A → D) → DS D E) → DS D E
```

$$[\![\_]\!] : \mathsf{DS}\ D\ E \to \mathsf{Fam}\ D \to \mathsf{Fam}\ E$$

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

Note: Fam $1 \cong$ Set and DS $1\ 1 \cong$ ID.

A code for a universe

The code

$$c_{\Sigma\mathbb{N}} = \sigma\ \{\mathsf{nat}, \mathsf{sig}\}\ (\mathsf{nat} \mapsto \iota\ \mathbb{N};$$
$$\mathsf{sig} \mapsto \delta\ 1\ (X \mapsto (\delta\ (X\,\star)\ (Y \mapsto \iota\ (\Sigma\ (X\,\star)\ Y)))))$$

represents $F : \mathsf{Fam\ Set} \to \mathsf{Fam\ Set}$ where

$$F(U, T) =$$
$$(1, \star \mapsto \mathbb{N}) + ((\Sigma s : U)(T s \to U), (s, p) \mapsto \Sigma\ (T\ s)\ (T \circ p))$$

A code for a universe

The code

$$c_{\Sigma\mathbb{N}} = \sigma\ \{\mathsf{nat}, \mathsf{sig}\}\ (\mathsf{nat} \mapsto \iota\ \mathbb{N};\ \mathsf{sig} \mapsto \delta\ 1\ (X \mapsto (\delta\ (X\,\star)\ (Y \mapsto \iota\ (\Sigma\ (X\,\star)\ Y)))))$$

represents $F : \mathsf{Fam\ Set} \to \mathsf{Fam\ Set}$ where

$$F(U,T) = (1, \star \mapsto \mathbb{N}) + \big((\Sigma s : U)(T s \to U), (s,p) \mapsto \Sigma\ (T\ s)\ (T \circ p)\big)$$

A code for a universe

The code

$$c_{\Sigma\mathbb{N}} = \sigma\ \{\mathsf{nat}, \mathsf{sig}\}\ (\mathsf{nat} \mapsto \iota\ \mathbb{N};$$
$$\mathsf{sig} \mapsto \delta\ 1\ (X \mapsto (\delta\ (X \star)\ (Y \mapsto \iota\ (\Sigma\ (X \star)\ Y)))))$$

represents $F : \mathsf{Fam\ Set} \to \mathsf{Fam\ Set}$ where

$$F(U, T) =$$
$$(1, \star \mapsto \mathbb{N}) + ((\Sigma s : 1 \to U)(T(s \star) \to U), (s, p) \mapsto \Sigma\ (T\ (s \star))\ (T \circ p))$$

A code for a universe

The code

$$c_{\Sigma\mathbb{N}} = \sigma\ \{\mathsf{nat}, \mathsf{sig}\}\ (\mathsf{nat} \mapsto \iota\ \mathbb{N};$$
$$\mathsf{sig} \mapsto \delta\ 1\ (X \mapsto (\delta\ (X\,\star)\ (Y \mapsto \iota\ (\Sigma\ (X\,\star)\ Y)))))$$

represents $F : \mathsf{Fam\ Set} \to \mathsf{Fam\ Set}$ where

$$F(U, T) =$$
$$(1, \star \mapsto \mathbb{N}) + ((\Sigma s : 1 \to U)(T(s\,\star) \to U), (s, p) \mapsto \Sigma\ (T\ (s\,\star))\ (T \circ p))$$

A code for a universe

The code

$$c_{\Sigma\mathbb{N}} = \sigma\ \{\mathsf{nat}, \mathsf{sig}\}\ (\mathsf{nat} \mapsto \iota\ \mathbb{N};$$
$$\mathsf{sig} \mapsto \delta\ 1\ (X \mapsto (\delta\ (X\,\star)\ (Y \mapsto \iota\ (\Sigma\ (X\,\star)\ Y)))))$$

represents $F : \mathsf{Fam\ Set} \to \mathsf{Fam\ Set}$ where

$$F(U, T) =$$
$$(1, \star \mapsto \mathbb{N}) + \big(\big(\Sigma s : 1 \to U\big)(T(s\,\star) \to U), (s, p) \mapsto \Sigma\ (T\ (s\,\star))\ (T \circ p)\big)$$

A code for a universe

The code

$$c_{\Sigma\mathbb{N}} = \sigma\ \{\mathsf{nat}, \mathsf{sig}\}\ (\mathsf{nat} \mapsto \iota\ \mathbb{N};\quad \mathsf{sig} \mapsto \delta\ 1\ (X \mapsto (\delta\ (X \star)\ (Y \mapsto \iota\ (\Sigma\ (X \star)\ Y)))))$$

represents $F : \mathsf{Fam\ Set} \to \mathsf{Fam\ Set}$ where

$$F(U, T) = (1, \star \mapsto \mathbb{N}) + \big((\Sigma s : 1 \to U)(T(s \star) \to U), (s, p) \mapsto \Sigma\ (T\ (s \star))\ (T \circ p)\big)$$

Closure under composition?

DS codes represent functors; are they closed under composition?

That is, given $c : \mathsf{DS}\ C\ D$ and $d : \mathsf{DS}\ D\ E$, is there a code $d \bullet c : \mathsf{DS}\ C\ E$ representing $[\![d]\!] \circ [\![c]\!] : \mathsf{Fam}\ C \to \mathsf{Fam}\ E$?

Closure under composition?

DS codes represent functors; are they closed under composition?

That is, given $c : \mathsf{DS}\ C\ D$ and $d : \mathsf{DS}\ D\ E$, is there a code $d \bullet c : \mathsf{DS}\ C\ E$ representing $[\![d]\!] \circ [\![c]\!] : \mathsf{Fam}\ C \to \mathsf{Fam}\ E$?

Why care?

Closure under composition?

DS codes represent functors; are they closed under composition?

That is, given $c : \mathsf{DS}\ C\ D$ and $d : \mathsf{DS}\ D\ E$, is there a code $d \bullet c : \mathsf{DS}\ C\ E$ representing $[\![d]\!] \circ [\![c]\!] : \mathsf{Fam}\ C \to \mathsf{Fam}\ E$?

Why care?

- Modularity: plug in c later.

Closure under composition?

DS codes represent functors; are they closed under composition?

That is, given $c : \mathsf{DS}\ C\ D$ and $d : \mathsf{DS}\ D\ E$, is there a code $d \bullet c : \mathsf{DS}\ C\ E$ representing $[\![d]\!] \circ [\![c]\!] : \mathsf{Fam}\ C \to \mathsf{Fam}\ E$?

Why care?

- Modularity: plug in c later.
- Solve $F(G(X)) \cong X$, not just $F(X) \cong X$. E.g. $c_{\mathsf{Rose}} = c_{\mathsf{List}} \bullet c_{\mathsf{List}}$.

Closure under composition?

DS codes represent functors; are they closed under composition?

That is, given $c : \mathsf{DS}\ C\ D$ and $d : \mathsf{DS}\ D\ E$, is there a code $d \bullet c : \mathsf{DS}\ C\ E$ representing $[\![d]\!] \circ [\![c]\!] : \mathsf{Fam}\ C \to \mathsf{Fam}\ E$?

Why care?

- Modularity: plug in c later.
- Solve $F(G(X)) \cong X$, not just $F(X) \cong X$. E.g. $c_{\mathsf{Rose}} = c_{\mathsf{List}} \bullet c_{\mathsf{List}}$.
- Longer term goal: want syntax-independent characterisation of induction-recursion (cf polynomial functors [Gambino and Kock]) — will likely be closed under composition.

A proof attempt

Define $d \bullet c$ by induction on d:

A proof attempt

Define $d \bullet c$ by induction on d:

Since $[\![\iota\ e]\!]\,([\![c]\!]\,(U, T)) = (1, \star \mapsto e)$,

$$(\iota\ e) \bullet c = \iota\ e$$

is easy.

A proof attempt

Define $d \bullet c$ by induction on d:

Since $[\![\iota\ e]\!]\,([\![c]\!]\,(U, T)) = (1, \star \mapsto e)$,

$$(\iota\ e) \bullet c = \iota\ e$$

is easy.

Similarly $(\sigma\ A\ f) \bullet c = \sigma\ A\ (a \mapsto (f\ a) \bullet d)$ by the induction hypothesis.

A proof attempt

Define $d \bullet c$ by induction on d:

Since $[\![\iota\ e]\!]\,([\![c]\!]\,(U, T)) = (1, \star \mapsto e)$,

$$(\iota\ e) \bullet c = \iota\ e$$

is easy.

Similarly $(\sigma\ A\ f) \bullet c = \sigma\ A\ (a \mapsto (f\ a) \bullet d)$ by the induction hypothesis.

But what about δ? (So far, we can compose with constant functors. . .)

Composing with δ

$$[\![\delta\ A\ F]\!]_0([\![c]\!]_0 Z) = \big(\Sigma g : A \to [\![c]\!]_0 Z\big)\big([\![F\big([\![c]\!]_1(Z) \circ g\big)]\!]_0([\![c]\!] Z)\big)$$

Composing with δ

$$[\![\delta\ A\ F]\!]_0([\![c]\!]_0 Z) = \left(\Sigma g : A \to [\![c]\!]_0 Z\right)\left([\![F\left([\![c]\!]_1(Z) \circ g\right)]\!]_0([\![c]\!] Z)\right)$$

Progress could be made if we had

1. $A \longrightarrow c$
2. "Concatenation" of codes

Composing with δ

$$[\![\delta\ A\ F]\!]_0([\![c]\!]_0 Z) = \left(\Sigma g : A \to [\![c]\!]_0 Z\right)\left([\![F\left([\![c]\!]_1(Z) \circ g\right)]\!]_0([\![c]\!] Z)\right)$$

Progress could be made if we had

1. $A \longrightarrow c$
2. "Concatenation" of codes

Spoiler alert: these are also necessary conditions.

“Concatenation” of codes

Item 2 is easy, because DS D is a monad (Ghani and Hancock [2016]):

Proposition. *There is an operation*

$$\_ \gg= \_ : DS\ C\ D \to (D \to DS\ C\ E) \to DS\ C\ E$$

such that $[\![c \gg= g]\!]\ Z \cong [\![c]\!]\ Z \gg=_{Fam} (e \mapsto [\![g\ e]\!]\ Z)$.

□

"Concatenation" of codes

Item 2 is easy, because DS D is a monad (Ghani and Hancock [2016]):

Proposition. *There is an operation*

$$\_ \gg= \_ : \mathit{DS}\ C\ D \to (D \to \mathit{DS}\ C\ E) \to \mathit{DS}\ C\ E$$

such that $[\![c \gg= g]\!]\ Z \cong [\![c]\!]\ Z \gg=_{\mathit{Fam}} (e \mapsto [\![g\ e]\!]\ Z)$.
Concretely,

$$[\![c \gg= g]\!]_0\ Z = \left(\Sigma x : [\![c]\!]_0\ Z\right) [\![g\,([\![c]\!]_1\ Z\,x)]\!]_0\ Z$$
$$[\![c \gg= g]\!]_1\ Z\,(x, y) = [\![g([\![c]\!]_1\ Z\,x)]\!]_1\ Z\,y$$

□

Trying to define $S \longrightarrow c$

This time ι and δ are easy, but:

$$\begin{aligned} S \to [\![\sigma\ A\ f]\!]_0\, Z &= S \to \big(\Sigma a : A\big)([\![f\ a]\!]_0\, Z) \\ &\cong \big(\Sigma g : S \to A\big)\big((x : S) \to [\![f\ (g\ x)]\!]_0\, Z\big) \end{aligned}$$

Trying to define $S \longrightarrow c$

This time ι and δ are easy, but:

$$
\begin{aligned}
S \to [\![\sigma\ A\ f]\!]_0\, Z &= S \to (\Sigma a : A)([\![f\ a]\!]_0\, Z) \\
&\cong (\Sigma g : S \to A)\big((x : S) \to [\![f\ (g\ x)]\!]_0\, Z\big)
\end{aligned}
$$

To continue inductively, we need to generalise to a dependent product

$$\pi : (S : \mathsf{Set}) \to (S \to \mathsf{DS}\ D\ E) \to \mathsf{DS}\ D\ E$$

Trying to define $S \longrightarrow c$

This time ι and δ are easy, but:

$$\begin{aligned} S \to [\![\sigma\ A\ f]\!]_0\ Z &= S \to \big(\Sigma a : A\big)([\![f\ a]\!]_0\ Z) \\ &\cong \big(\Sigma g : S \to A\big)\big((x : S) \to [\![f\ (g\ x)]\!]_0\ Z\big) \end{aligned}$$

To continue inductively, we need to generalise to a dependent product

$$\pi : \big(S : \mathsf{Set}\big) \to (S \to \mathsf{DS}\ D\ E) \to \mathsf{DS}\ D\ E$$

But we cannot define this because we have nothing to induct on anymore.

Powers from composition

In fact, any definition of composition would give us powers:

Theorem. *A composition operator*

$$\_ \bullet \_ : DS\ D\ E \to DS\ C\ D \to DS\ C\ E$$

is definable if and only if a power operator

$$\_ \longrightarrow \_ : (S : Set) \to DS\ D\ E \to DS\ D\ (S \to E)$$

is definable. □

Powers from composition

In fact, any definition of composition would give us powers:

Theorem. *A composition operator*

$$\_ \bullet \_ : \mathit{DS}\ D\ E \to \mathit{DS}\ C\ D \to \mathit{DS}\ C\ E$$

is definable if and only if a power operator

$$\_ \longrightarrow \_ : (S : \mathit{Set}) \to \mathit{DS}\ D\ E \to \mathit{DS}\ D\ (S \to E)$$

is definable. □

This (apparent) lack of powers thus suggests that DS, as an axiomatisation of a class of functors, could perhaps be improved upon.

Variations on inductive-recursive definitions

This leads us to investigate alternative classes of functors axiomatising inductive-recursive definitions.

If one wants closure under composition, two natural options suggest themselves:

1. Restrict dependency so that $S \longrightarrow c$ is definable $\rightsquigarrow$ uniform codes (Peter Hancock).

2. Add a π combinator to the system $\rightsquigarrow$ polynomial codes (Conor McBride).

Variations on inductive-recursive definitions

This leads us to investigate alternative classes of functors axiomatising inductive-recursive definitions.

If one wants closure under composition, two natural options suggest themselves:

1. Restrict dependency so that $S \longrightarrow c$ is definable $\rightsquigarrow$ uniform codes (Peter Hancock).
2. Add a π combinator to the system $\rightsquigarrow$ polynomial codes (Conor McBride).

Take-home message: There are many axiomatisations of induction-recursion.

Uniform codes

Uniform codes

Originally due to Peter Hancock (2012).

Discovered while trying to define composition for DS.

Uniformity by associating like in the 60s

In

$$\sigma : (A : \mathsf{Set}) \to (c : A \to \mathsf{DS}\,DE) \to \mathsf{DS}\,DE$$

nonuniformity comes from c depending on A.

Uniformity by associating like in the 60s

In

$$\sigma : (A : \mathsf{Set}) \to (c : A \to \mathsf{DS}\,DE) \to \mathsf{DS}\,DE$$

nonuniformity comes from c depending on A.

Idea: Instead make A depend on (the information in) c.

Uniformity by associating like in the 60s

In

$$\sigma : (A : \mathsf{Set}) \to (c : A \to \mathsf{DS}\,D\,E) \to \mathsf{DS}\,D\,E$$

nonuniformity comes from c depending on A.

Idea: Instead make A depend on (the information in) c.

Consequence: the code c for "the rest of the constructor" is always of the same "shape".

Uniformity by associating like in the 60s

In

$$\sigma : (A : \mathsf{Set}) \to (c : A \to \mathsf{DS}\,DE) \to \mathsf{DS}\,DE$$

nonuniformity comes from c depending on A.

Idea: Instead make A depend on (the information in) c.

Consequence: the code c for "the rest of the constructor" is always of the same "shape".

Left-nested instead of right-nested (Pollack: Dependently Typed Records in Type Theory [2002]).

Uniform codes UF

Let $D, E : \mathsf{Set}_1$. $\mathsf{Uni}\ D : \mathsf{Set}_1$ and $\mathsf{Info} : \mathsf{Uni}\ D \to \mathsf{Set}_1$ are inductive-recursively given by

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

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

Large set of uniform codes $\mathsf{UF}\ D\ E = (\Sigma c : \mathsf{Uni}\ D)(\mathsf{Info}\ c \to E)$.

Decoding uniform codes

$$[\![\_]\!]_{\mathsf{Uni}} : \mathsf{Uni}\ D \to \mathsf{Fam}\ D \to \mathsf{Set}$$

$$[\![\_]\!]_{\mathsf{Info}} : (c : \mathsf{Uni}\ D) \to (Z : \mathsf{Fam}\ D) \to [\![\ c\ ]\!]_{\mathsf{Uni}}\ Z \to \mathsf{Info}\ c$$

Decoding uniform codes

$$\llbracket \_ \rrbracket_{\mathsf{Uni}} : \mathsf{Uni}\ D \to \mathsf{Fam}\ D \to \mathsf{Set}$$
$$\llbracket \_ \rrbracket_{\mathsf{Info}} : (c : \mathsf{Uni}\ D) \to (Z : \mathsf{Fam}\ D) \to \llbracket c \rrbracket_{\mathsf{Uni}}\ Z \to \mathsf{Info}\ c$$

$$\llbracket \iota_{\mathsf{UF}} \rrbracket_{\mathsf{Uni}}\ (U, T) = 1$$
$$\llbracket \sigma_{\mathsf{UF}}\ c\ A \rrbracket_{\mathsf{Uni}}\ (U, T) = \big(\Sigma x : \llbracket c \rrbracket_{\mathsf{Uni}}\ (U, T)\big)(A(\llbracket c \rrbracket_{\mathsf{Info}}\ (U, T)\ x))$$
$$\llbracket \delta_{\mathsf{UF}}\ c\ A \rrbracket_{\mathsf{Uni}}\ (U, T) = \big(\Sigma x : \llbracket c \rrbracket_{\mathsf{Uni}}\ (U, T)\big)(A(\llbracket c \rrbracket_{\mathsf{Info}}\ (U, T)\ x) \to U)$$

$$\vdots$$

$$\llbracket \delta_{\mathsf{UF}}\ c\ S \rrbracket_{\mathsf{Info}}\ (U, T)\ (x, g) = (\llbracket c \rrbracket_{\mathsf{Info}}\ (U, T)\ x, T \circ g)$$

Decoding uniform codes

$$\llbracket \_ \rrbracket_{\mathsf{Uni}} : \mathsf{Uni}\ D \to \mathsf{Fam}\ D \to \mathsf{Set}$$
$$\llbracket \_ \rrbracket_{\mathsf{Info}} : (c : \mathsf{Uni}\ D) \to (Z : \mathsf{Fam}\ D) \to \llbracket c \rrbracket_{\mathsf{Uni}}\ Z \to \mathsf{Info}\ c$$

$$\llbracket \iota_{\mathsf{UF}} \rrbracket_{\mathsf{Uni}}\ (U, T) = 1$$
$$\llbracket \sigma_{\mathsf{UF}}\ c\ A \rrbracket_{\mathsf{Uni}}\ (U, T) = (\Sigma x : \llbracket c \rrbracket_{\mathsf{Uni}}\ (U, T))(A(\llbracket c \rrbracket_{\mathsf{Info}}\ (U, T)\, x))$$
$$\llbracket \delta_{\mathsf{UF}}\ c\ A \rrbracket_{\mathsf{Uni}}\ (U, T) = (\Sigma x : \llbracket c \rrbracket_{\mathsf{Uni}}\ (U, T))(A(\llbracket c \rrbracket_{\mathsf{Info}}\ (U, T)\, x) \to U)$$

$$\vdots$$

$$\llbracket \delta_{\mathsf{UF}}\ c\ S \rrbracket_{\mathsf{Info}}\ (U, T)\ (x, g) = (\llbracket c \rrbracket_{\mathsf{Info}}\ (U, T)\ x, T \circ g)$$

Finally for $(c, \alpha) : \mathsf{UF}\ D\ E = (\Sigma c : \mathsf{Uni}\ D)(\mathsf{Info}\ c \to E)$

$$\llbracket (c, \alpha) \rrbracket = (\llbracket c \rrbracket_{\mathsf{Uni}}\ -, \alpha \circ \llbracket c \rrbracket_{\mathsf{Info}}\ -) : \mathsf{Fam}\ D \to \mathsf{Fam}\ E$$

A code for W-types

```
data W (S : Set)(P : S → Set) : Set where
  sup : (s : S) → (P s → W S P) → W S P
```

$$c_{\mathsf{W}\ S\ P,\mathsf{UF}} = \delta_{\mathsf{UF}}\ \big(\sigma_{\mathsf{UF}}\ \iota_{\mathsf{UF}}\ (\_ \mapsto S)\big)\ ((\_, s) \mapsto (P\ s)) : \mathsf{Uni}\ 1$$

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

A code for W-types

```
data W (S : Set)(P : S → Set) : Set where
  sup : (s : S) → (P s → W S P) → W S P
```

$$c_{\mathsf{W}\ S\ P,\mathsf{UF}} = \delta_{\mathsf{UF}}\ \big(\sigma_{\mathsf{UF}}\ \iota_{\mathsf{UF}}\ (\_ \mapsto S)\big)\ ((\_, s) \mapsto (P\,s)) : \mathsf{Uni}\ 1$$

$$c_{\mathsf{W}\ S\ P,\mathsf{DS}} = \sigma\, S\, (s \mapsto \delta\, (P\,s)\, (\_ \mapsto \iota\, \star)) : \mathsf{DS}\ 1\ 1$$

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

A code for W-types

```
data W (S : Set)(P : S → Set) : Set where
  sup: (s : S) → (P s → W S P) → W S P
```

$$c_{W\,S\,P,\mathrm{UF}} = \delta_{\mathrm{UF}}\,(\sigma_{\mathrm{UF}}\;\iota_{\mathrm{UF}}\;(\_ \mapsto S))\;((\_, s) \mapsto (P\,s)) : \mathrm{Uni}\;1$$

$$c_{W\,S\,P,\mathrm{DS}} = \sigma\,S\,(s \mapsto \delta\,(P\,s)\,(\_ \mapsto \iota\,\star)) : \mathrm{DS}\;1\;1$$

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

A code for W-types

```
data W (S : Set)(P : S → Set) : Set where
  sup : (s : S) → (P s → W S P) → W S P
```

$$c_{\mathsf{W}\ S\ P,\mathsf{UF}} = \delta_{\mathsf{UF}}\ \big(\sigma_{\mathsf{UF}}\ \iota_{\mathsf{UF}}\ (\_ \mapsto S)\big)\ ((\_, s) \mapsto (P\ s)) : \mathsf{Uni}\ 1$$

$$c_{\mathsf{W}\ S\ P,\mathsf{DS}} = \sigma\ S\ (s \mapsto \delta\ (P\ s)\ (\_ \mapsto \iota\ \star)) : \mathsf{DS}\ 1\ 1$$

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

A code for W-types

```
data W (S : Set)(P : S → Set) : Set where
  sup: (s : S) → (P s → W S P) → W S P
```

$$c_{\mathsf{W}\ S\ P,\mathsf{UF}} = \delta_{\mathsf{UF}}\ \big(\sigma_{\mathsf{UF}}\ \iota_{\mathsf{UF}}\ (\_ \mapsto S)\big)\ ((\_, s) \mapsto (P\ s)) : \mathsf{Uni}\ 1$$
$$c_{\mathsf{W}\ S\ P,\mathsf{DS}} = \sigma\ S\ (s \mapsto \delta\ (P\ s)\ (\_ \mapsto \iota\ \star)) : \mathsf{DS}\ 1\ 1$$

$$[\![\ c_{\mathsf{W}\ S\ P,\mathsf{UF}}\ ]\!]_{\mathsf{Uni}}\ (U, T) = \big(\Sigma(\star, s) : 1 \times S\big)(P(s) \to U)$$
$$[\![c_{\mathsf{W}\ S\ P,\mathsf{DS}}]\!]_0\ (U, T) = \big(\Sigma s : S\big)\big(\Sigma f : (P(s) \to U)\big)1$$

Coproducts of uniform codes

A priori we do not longer have coproducts of codes — DS coproducts relied exactly on non-uniformity of σ.

Coproducts of uniform codes

A priori we do not longer have coproducts of codes — DS coproducts relied exactly on non-uniformity of σ.

Crucial for encoding several constructors into one.

Coproducts of uniform codes

A priori we do not longer have coproducts of codes — DS coproducts relied exactly on non-uniformity of σ.

Crucial for encoding several constructors into one.

Proposition. *For every uniform code c,* $[\![c]\!] Z \cong [\![\sigma_{\mathsf{UF}}\, c\, (\_ \mapsto 1)]\!] Z$ *and* $[\![c]\!] Z \cong [\![\delta_{\mathsf{UF}}\, c\, (\_ \mapsto 0)]\!] Z$. □

By "padding" codes with such semantically redundant information, we can define $c +_{\mathsf{UF}} d$.

Coproducts of uniform codes

A priori we do not longer have coproducts of codes — DS coproducts relied exactly on non-uniformity of σ.

Crucial for encoding several constructors into one.

Proposition. *For every uniform code c,* $[\![\, c \,]\!]\, Z \cong [\![\, \sigma_{\mathsf{UF}}\, c\, (\_ \mapsto 1) \,]\!]\, Z$ *and* $[\![\, c \,]\!]\, Z \cong [\![\, \delta_{\mathsf{UF}}\, c\, (\_ \mapsto 0) \,]\!]\, Z$. □

By "padding" codes with such semantically redundant information, we can define $c +_{\mathsf{UF}} d$.

E.g.

$$\sigma_{\mathsf{UF}}\, (\delta_{\mathsf{UF}}\, \iota_{\mathsf{UF}}\, A)\, B +_{\mathsf{UF}} \delta_{\mathsf{UF}}\, \iota_{\mathsf{UF}}\, A' = \sigma_{\mathsf{UF}}\, (\delta_{\mathsf{UF}}\, (\sigma_{\mathsf{UF}}\, \iota_{\mathsf{UF}}\, 2)\, [A, A'])\, [B, 0]$$

Coproducts of uniform codes

A priori we do not longer have coproducts of codes — DS coproducts relied exactly on non-uniformity of σ.

Crucial for encoding several constructors into one.

Proposition. *For every uniform code* c, $[\![c]\!] Z \cong [\![\sigma_{\mathsf{UF}}\, c\, (\_ \mapsto 1)]\!] Z$ *and* $[\![c]\!] Z \cong [\![\delta_{\mathsf{UF}}\, c\, (\_ \mapsto 0)]\!] Z$. □

By "padding" codes with such semantically redundant information, we can define $c +_{\mathsf{UF}} d$.

E.g.

$$\sigma_{\mathsf{UF}}\,(\delta_{\mathsf{UF}}\,\iota_{\mathsf{UF}}\,A)\,B +_{\mathsf{UF}} \delta_{\mathsf{UF}}\,\iota_{\mathsf{UF}}\,A' = \sigma_{\mathsf{UF}}\,(\delta_{\mathsf{UF}}\,(\sigma_{\mathsf{UF}}\,\iota_{\mathsf{UF}}\,2)\,[A, A'])\,[B, 0]$$

Theorem. $[\![c +_{\mathsf{UF}} d]\!] Z \cong [\![c]\!] Z + [\![d]\!] Z$. □

UF $\hookrightarrow$ DS

Since uniform codes are "backwards", we can translate UF to DS the same way one reverses a list using an accumulator:

$$\mathsf{accUFtoDS} : (c : \mathsf{Uni}\ D) \to (\mathsf{Info}\ c \to \mathsf{DS}\ D\ E) \to \mathsf{DS}\ D\ E$$

UF $\hookrightarrow$ DS

Since uniform codes are “backwards”, we can translate UF to DS the same way one reverses a list using an accumulator:

$$\mathsf{accUFtoDS} : (c : \mathsf{Uni}\ D) \to (\mathsf{Info}\ c \to \mathsf{DS}\ D\ E) \to \mathsf{DS}\ D\ E$$

defined by

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

UF $\hookrightarrow$ DS

Since uniform codes are "backwards", we can translate UF to DS the same way one reverses a list using an accumulator:

$$\mathsf{accUFtoDS} : (c : \mathsf{Uni}\ D) \to (\mathsf{Info}\ c \to \mathsf{DS}\ D\ E) \to \mathsf{DS}\ D\ E$$

defined by

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

Proposition. $[\![\mathit{accUFtoDS}\ c\ (\iota \circ \alpha)]\!]\ Z \cong [\![\ (c, \alpha)\ ]\!]\ Z.$ □

Going the other way seems unlikely.

Consequences for soundness

This means that UF can piggyback on Dybjer and Setzer [1999]'s proof of existence of initial algebras.

Consequences for soundness

This means that UF can piggyback on Dybjer and Setzer [1999]'s proof of existence of initial algebras.

However the construction of (Uni, Info) itself is one instance of large induction-recursion, albeit a particularly simple instance. No additional assumptions are needed in the set-theoretical model.

UF is not a monad

We have gained uniformity, which makes powers definable.

Unfortunately, the uniformity also means that we no longer have a monad.

UF is not a monad

We have gained uniformity, which makes powers definable.

Unfortunately, the uniformity also means that we no longer have a monad.

Bind should graft trees, but grafting a collection of uniform trees might not result in a uniform tree.

Towards composition: combined bind and powers

Is all lost? No. We can still define the instance of bind that we need, combined with a power operation. (Note: only the set depends on Info c.)

$$- \gg\!\!=\![- \longrightarrow -] : (c : \mathsf{Uni}\ D) \to (\mathsf{Info}\ c \to \mathsf{Set}) \to \mathsf{Uni}\ D \to \mathsf{Uni} D$$

Towards composition: combined bind and powers

Is all lost? No. We can still define the instance of bind that we need, combined with a power operation. (Note: only the set depends on Info c.)

$$- \gg\!\!= [- \longrightarrow -] : (c : \mathsf{Uni}\ D) \to (\mathsf{Info}\ c \to \mathsf{Set}) \to \mathsf{Uni}\ D \to \mathsf{Uni} D$$

As usual, we need to define this simultaneously with its meaning on Info:

$$(c \gg\!\!= [E \longrightarrow d])_{\mathsf{Info}} : \mathsf{Info}\ (c \gg\!\!= [E \longrightarrow d]) \to \big(\Sigma x : \mathsf{Info}\ c\big)(E\, x \to \mathsf{Info}\ d)$$

Towards composition: combined bind and powers

Is all lost? No. We can still define the instance of bind that we need, combined with a power operation. (Note: only the set depends on Info c.)

$$- \gg\!\!=\![- \longrightarrow -] : (c : \mathsf{Uni}\ D) \to (\mathsf{Info}\ c \to \mathsf{Set}) \to \mathsf{Uni}\ D \to \mathsf{Uni} D$$

As usual, we need to define this simultaneously with its meaning on Info:

$$(c \gg\!\!=\![E \longrightarrow d])_{\mathsf{Info}} : \mathsf{Info}\ (c \gg\!\!=\![E \longrightarrow d]) \to \big(\Sigma x : \mathsf{Info}\ c\big)(E\, x \to \mathsf{Info}\ d)$$

Proposition. *There is an equivalence*

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

Composition for UF

$$\_ \bullet_{\mathsf{Uni}} \_ : \mathsf{Uni}\ D \to \mathsf{UF}\ C\ D \to \mathsf{Uni}\ C$$

$$(\_ \bullet_{\mathsf{Info}} \_) : (c : \mathsf{Uni}\ D) \to (R : \mathsf{UF}\ C\ D) \to \mathsf{Info}\ (c \bullet_{\mathsf{Uni}} R) \to \mathsf{Info}\ c$$

Composition for UF

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

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

Composition for UF

$$\_ \bullet_{\mathsf{Uni}} \_ : \mathsf{Uni}\ D \to \mathsf{UF}\ C\ D \to \mathsf{Uni}\ C$$
$$(\_ \bullet_{\mathsf{Info}} \_) : (c : \mathsf{Uni}\ D) \to (R : \mathsf{UF}\ C\ D) \to \mathsf{Info}\ (c \bullet_{\mathsf{Uni}} R) \to \mathsf{Info}\ c$$

$$\iota_{\mathsf{UF}} \bullet_{\mathsf{Uni}} R = \iota_{\mathsf{UF}}$$
$$(\sigma_{\mathsf{UF}}\ c\ A) \bullet_{\mathsf{Uni}} R = \sigma_{\mathsf{UF}}\ (c \bullet_{\mathsf{Uni}} R)\ (A \circ (c \bullet_{\mathsf{Info}} R))$$
$$(\delta_{\mathsf{UF}}\ c\ A) \bullet_{\mathsf{Uni}} (d, \beta) = (c \bullet_{\mathsf{Uni}} (d, \beta)) \gg\!= [(A \circ (c \bullet_{\mathsf{Info}} (d, \beta))) \longrightarrow d]$$

Theorem.
$\llbracket\, (c, \alpha) \bullet d \,\rrbracket\, Z = \llbracket\, c \bullet_{Uni} d, \alpha \circ (c \bullet_{Info}\ d) \,\rrbracket\, Z \cong \llbracket\, (c, \alpha) \,\rrbracket (\llbracket\, d \,\rrbracket\, Z).$ □

How suitable are uniform codes?

Uniform codes (most likely) capture a smaller class of functors compared to DS.

However all inductive-recursive definitions "in the wild" are already uniform (because coproducts definable).

How suitable are uniform codes?

Uniform codes (most likely) capture a smaller class of functors compared to DS.

However all inductive-recursive definitions “in the wild” are already uniform (because coproducts definable).

Conjecture: UF and DS have the same proof-theoretical strength.

Summary

Uniform codes UF and polynomial codes PN as new, alternative axiomatisations of inductive-recursive definitions.

$$\mathrm{UF} \hookrightarrow \mathrm{DS} \hookrightarrow \mathrm{PN}$$

Both UF and PN are closed under composition; DS probably is not.

Existence of initial algebras for UF unproblematic. For PN, need to adjust the DS model slightly (but not much).

Are there other, even more well-behaved axiomatisations?

Thank you!