

A Formal Comparison of Approaches to Datatype-Generic Programming

José Pedro Magalhães
joint work with Andres Löh

Summer School on Metaprogramming
Cambridge, UK

August 8, 2016

Outline

Introduction

Generic programming libraries, in Agda

Regular

PolyP

Indexed functors

Instant generics

Conversions

Conclusion

Setting

- ▶ There are *many* libraries for generic programming in Haskell
- ▶ Different approaches vary wildly in what datatypes they can encode (universe size) and in what functionality they can offer (expressiveness)
- ▶ There is a lot of duplicated code across different libraries
- ▶ Newcomers to the field never know what library to use
- ▶ Informal comparisons exist, but there are no embeddings, nor formalised statements

Setting

- ▶ There are *many* libraries for generic programming in Haskell
- ▶ Different approaches vary wildly in what datatypes they can encode (universe size) and in what functionality they can offer (expressiveness)
- ▶ There is a lot of duplicated code across different libraries
- ▶ Newcomers to the field never know what library to use
- ▶ Informal comparisons exist, but there are no embeddings, nor formalised statements

We intend to change this.

Generic programming libraries, in Agda

We have looked at five libraries:

- ▶ `regular`: simple library, one recursive position, no parameters

Generic programming libraries, in Agda

We have looked at five libraries:

- ▶ `regular`: simple library, one recursive position, no parameters
- ▶ `polyp`: historical approach, one recursive position, one parameter, and composition

Generic programming libraries, in Agda

We have looked at five libraries:

- ▶ `regular`: simple library, one recursive position, no parameters
- ▶ `polyp`: historical approach, one recursive position, one parameter, and composition
- ▶ `multirec`: multiple recursive positions, no parameters (omitted from this talk)

Generic programming libraries, in Agda

We have looked at five libraries:

- ▶ `regular`: simple library, one recursive position, no parameters
- ▶ `polyp`: historical approach, one recursive position, one parameter, and composition
- ▶ `multirec`: multiple recursive positions, no parameters (omitted from this talk)
- ▶ `indexed`: multiple recursive positions, multiple parameters, composition, and fixed points within the universe

Generic programming libraries, in Agda

We have looked at five libraries:

- ▶ `regular`: simple library, one recursive position, no parameters
- ▶ `polyp`: historical approach, one recursive position, one parameter, and composition
- ▶ `multirec`: multiple recursive positions, no parameters (omitted from this talk)
- ▶ `indexed`: multiple recursive positions, multiple parameters, composition, and fixed points within the universe
- ▶ `instant-generics`: coinductive approach with recursive codes

Outline

Introduction

Generic programming libraries, in Agda

Regular

PolyP

Indexed functors

Instant generics

Conversions

Conclusion

Regular—universe

```
module Regular where
  data Code : Set where
    U       : Code
    I       : Code
    _⊕_     : (F G : Code) → Code
    _⊗_     : (F G : Code) → Code
```

Regular—interpretation

$\llbracket _ \rrbracket : \text{Code} \rightarrow (\text{Set} \rightarrow \text{Set})$

$\llbracket \text{U} \rrbracket A = \top$

$\llbracket \text{I} \rrbracket A = A$

$\llbracket F \oplus G \rrbracket A = \llbracket F \rrbracket A \uplus \llbracket G \rrbracket A$

$\llbracket F \otimes G \rrbracket A = \llbracket F \rrbracket A \times \llbracket G \rrbracket A$

data μ (F : Code) : Set where

$\langle _ \rangle : \llbracket F \rrbracket (\mu F) \rightarrow \mu F$

Regular—map

$\text{map} : \{A\ B : \text{Set}\} (F : \text{Code})$
 $\rightarrow (A \rightarrow B) \rightarrow \llbracket F \rrbracket A \rightarrow \llbracket F \rrbracket B$

$\text{map } U \quad f _ \quad = \text{tt}$

$\text{map } I \quad f\ x \quad = f\ x$

$\text{map } (F \oplus G) f (\text{inj}_1\ x) = \text{inj}_1 (\text{map } F\ f\ x)$

$\text{map } (F \oplus G) f (\text{inj}_2\ x) = \text{inj}_2 (\text{map } G\ f\ x)$

$\text{map } (F \otimes G) f (x, y) = \text{map } F\ f\ x, \text{map } G\ f\ y$

Outline

Introduction

Generic programming libraries, in Agda

Regular

PolyP

Indexed functors

Instant generics

Conversions

Conclusion

PolyP—universe

```
module PolyP where
```

```
  data Code : Set where
```

```
    U      : Code
```

```
    I      : Code
```

```
    P      : Code
```

```
    _⊕_    : (F G : Code) → Code
```

```
    _⊗_    : (F G : Code) → Code
```

```
    _⊙_    : (F G : Code) → Code
```

PolyP—interpretation

mutual

$\llbracket _ \rrbracket : \text{Code} \rightarrow (\text{Set} \rightarrow \text{Set} \rightarrow \text{Set})$

$\llbracket \text{U} \rrbracket A R = \top$

$\llbracket \text{I} \rrbracket A R = R$

$\llbracket \text{P} \rrbracket A R = A$

$\llbracket \text{F} \oplus \text{G} \rrbracket A R = \llbracket \text{F} \rrbracket A R \uplus \llbracket \text{G} \rrbracket A R$

$\llbracket \text{F} \otimes \text{G} \rrbracket A R = \llbracket \text{F} \rrbracket A R \times \llbracket \text{G} \rrbracket A R$

$\llbracket \text{F} \odot \text{G} \rrbracket A R = \mu F (\llbracket \text{G} \rrbracket A R)$

data $\mu (F : \text{Code}) (A : \text{Set}) : \text{Set}$ where

$\langle _ \rangle : \llbracket F \rrbracket A (\mu F A) \rightarrow \mu F A$

PolyP—map

mutual

map : {A B R S : Set} (F : Code)
→ (A → B) → (R → S) → [[F]] A R → [[F]] B S

map U f g _ = tt

map I f g x = g x

map P f g x = f x

map (F ⊕ G) f g (inj₁ x) = inj₁ (map F f g x)

map (F ⊕ G) f g (inj₂ x) = inj₂ (map G f g x)

map (F ⊗ G) f g (x, y) = map F f g x, map G f g y

map (F ⊙ G) f g ⟨ x ⟩ = ⟨ map F (map G f g)
(map (F ⊙ G) f g) x ⟩

pmap : {A B : Set} (F : Code)

→ (A → B) → μ F A → μ F B

pmap F f ⟨ x ⟩ = ⟨ map F f (pmap F f) x ⟩

Outline

Introduction

Generic programming libraries, in Agda

Regular

PolyP

Indexed functors

Instant generics

Conversions

Conclusion

Indexed functors—universe

module Indexed **where**

data Code (I : Set) (O : Set) : Set **where**

U : Code I O

I : I → Code I O

! : O → Code I O

$_ \oplus _$: (F G : Code I O) → Code I O

$_ \otimes _$: (F G : Code I O) → Code I O

$_ \odot _$: {M : Set} → (F : Code M O)
→ (G : Code I M) → Code I O

Fix : (F : Code (I \uplus O) O) → Code I O

Indexed functors—interpretation

Indexed : Set \rightarrow Set

Indexed I = I \rightarrow Set

mutual

$\llbracket _ \rrbracket : \{ I O : \text{Set} \} \rightarrow \text{Code } I O \rightarrow \text{Indexed } I \rightarrow \text{Indexed } O$

$\llbracket U \rrbracket r_i = \top$

$\llbracket I_j \rrbracket r_i = r_j$

$\llbracket !j \rrbracket r_i = i \equiv j$

$\llbracket F \oplus G \rrbracket r_i = \llbracket F \rrbracket r_i \uplus \llbracket G \rrbracket r_i$

$\llbracket F \otimes G \rrbracket r_i = \llbracket F \rrbracket r_i \times \llbracket G \rrbracket r_i$

$\llbracket F \odot G \rrbracket r_i = \llbracket F \rrbracket (\llbracket G \rrbracket r)_i$

$\llbracket \text{Fix } F \rrbracket r_i = \mu F r_i$

data $\mu \{ I O : \text{Set} \} (F : \text{Code } (I \uplus O) O)$

(r : Indexed I) (o : O) : Set **where**

$\langle _ \rangle : \llbracket F \rrbracket [r, \mu F r] o \rightarrow \mu F r o$

Indexed functors—map

$$\begin{aligned} \text{map} &: \{I O : \text{Set}\} \{R S : \text{Indexed } I\} (F : \text{Code } I O) \\ &\rightarrow R \Rightarrow S \rightarrow \llbracket F \rrbracket R \Rightarrow \llbracket F \rrbracket S \\ \text{map } U &\quad f i _ \quad = \text{tt} \\ \text{map } (l j) &\quad f i x \quad = f j x \\ \text{map } (! j) &\quad f i x \quad = x \\ \text{map } (F \oplus G) & f i (\text{inj}_1 x) = \text{inj}_1 (\text{map } F f i x) \\ \text{map } (F \oplus G) & f i (\text{inj}_2 x) = \text{inj}_2 (\text{map } G f i x) \\ \text{map } (F \otimes G) & f i (x, y) = \text{map } F f i x, \text{map } G f i y \\ \text{map } (F \odot G) & f i x \quad = \text{map } F (\text{map } G f) i x \\ \text{map } (\text{Fix } F) & f i \langle x \rangle = \langle \text{map } F (f \parallel \text{map } (\text{Fix } F) f) i x \rangle \end{aligned}$$

Indexed functors—map

$$_ \rightrightarrows _ : \{I : \text{Set}\} \rightarrow \text{Indexed } I \rightarrow \text{Indexed } I \rightarrow \text{Set}$$
$$R \rightrightarrows S = (i : _) \rightarrow R\ i \rightarrow S\ i$$
$$_ \parallel _ : \{I\ J : \text{Set}\} \{A\ C : \text{Indexed } I\} \{B\ D : \text{Indexed } J\}$$
$$\rightarrow A \rightrightarrows C \rightarrow B \rightrightarrows D \rightarrow [A, B] \rightrightarrows [C, D]$$
$$\text{map} : \{I\ O : \text{Set}\} \{R\ S : \text{Indexed } I\} (F : \text{Code } I\ O)$$
$$\rightarrow R \rightrightarrows S \rightarrow \llbracket F \rrbracket R \rightrightarrows \llbracket F \rrbracket S$$
$$\text{map } U \quad f\ i\ _ \quad = \text{tt}$$
$$\text{map } (I\ j) \quad f\ i\ x \quad = f\ j\ x$$
$$\text{map } (!\ j) \quad f\ i\ x \quad = x$$
$$\text{map } (F \oplus G) \quad f\ i\ (\text{inj}_1\ x) = \text{inj}_1\ (\text{map } F\ f\ i\ x)$$
$$\text{map } (F \oplus G) \quad f\ i\ (\text{inj}_2\ x) = \text{inj}_2\ (\text{map } G\ f\ i\ x)$$
$$\text{map } (F \otimes G) \quad f\ i\ (x, y) = \text{map } F\ f\ i\ x, \text{map } G\ f\ i\ y$$
$$\text{map } (F \odot G) \quad f\ i\ x \quad = \text{map } F\ (\text{map } G\ f)\ i\ x$$
$$\text{map } (\text{Fix } F) \quad f\ i\ \langle x \rangle \quad = \langle \text{map } F\ (f \parallel \text{map } (\text{Fix } F)\ f)\ i\ x \rangle$$

Outline

Introduction

Generic programming libraries, in Agda

Regular

PolyP

Indexed functors

Instant generics

Conversions

Conclusion

Instant generics—universe

```
module InstantGenerics where
```

```
mutual
```

```
data Code : Set where
```

```
  U      :                               Code
```

```
  K      : Set                            → Code
```

```
  R      : ∞ Code                         → Code
```

```
  _⊕_    : Code → Code → Code
```

```
  _⊗_    : Code → Code → Code
```


Instant generics—interpretation

data $\llbracket _ \rrbracket$: **Code** \rightarrow **Set** **where**

tt : $\llbracket \text{U} \rrbracket$

k : $\{A : \text{Set}\} \rightarrow A \rightarrow \llbracket \text{K } A \rrbracket$

rec : $\{C : \infty \text{ Code}\} \rightarrow \llbracket \text{b } C \rrbracket \rightarrow \llbracket \text{R } C \rrbracket$

inl : $\{C D : \text{Code}\} \rightarrow \llbracket C \rrbracket \rightarrow \llbracket C \oplus D \rrbracket$

inr : $\{C D : \text{Code}\} \rightarrow \llbracket D \rrbracket \rightarrow \llbracket C \oplus D \rrbracket$

_ , _ : $\{C D : \text{Code}\} \rightarrow \llbracket C \rrbracket \rightarrow \llbracket D \rrbracket \rightarrow \llbracket C \otimes D \rrbracket$

Instant generics—sample generic function

Coinductive codes do not naturally define functors. We show an example generic function:

$$\begin{aligned} \text{size} &: (A : \text{Code}) \rightarrow \llbracket A \rrbracket \rightarrow \mathbb{N} \\ \text{size } U \quad x &= 1 \\ \text{size } (K A) \quad (k x) &= 1 \\ \text{size } (R C) \quad (\text{rec } x) &= 1 + \text{size } (b C) x \\ \text{size } (A \oplus B) \quad (\text{inl } x) &= \text{size } A x \\ \text{size } (A \oplus B) \quad (\text{inr } x) &= \text{size } B x \\ \text{size } (A \otimes B) \quad (x, y) &= \text{size } A x + \text{size } B y \end{aligned}$$

Encoding datatypes

module Example **where**

data List (A : Set) : Set **where**

nil : List A

cons : A → List A → List A

ListC_p : Code_p

ListC_p = U_p ⊕_p P_p ⊗_p I_p

ListC_i : Code_i T T

ListC_i = Fix_i (U_i ⊕_i (I_i (inj₁ tt)) ⊗_i (I_i (inj₂ tt)))

ListC_{ig} : Code_{ig} → Code_{ig}

ListC_{ig} A = U_{ig} ⊕_{ig} (A ⊗_{ig} (R_{ig} (# (ListC_{ig} A))))

Outline

Introduction

Generic programming libraries, in Agda

- Regular

- PolyP

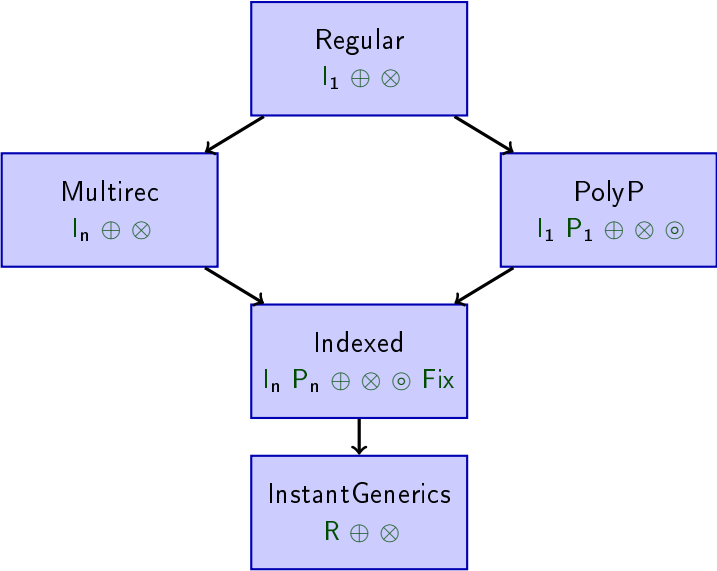
- Indexed functors

- Instant generics

Conversions

Conclusion

Conversions



Regular to PolyP I

Code conversion:

$$\uparrow^P : \text{Code}_r \rightarrow \text{Code}_p$$

$$\uparrow^P U_r = U_p$$

$$\uparrow^P I_r = I_p$$

$$\uparrow^P (F \oplus_r G) = (\uparrow^P F) \oplus_p (\uparrow^P G)$$

$$\uparrow^P (F \otimes_r G) = (\uparrow^P F) \otimes_p (\uparrow^P G)$$

Regular to PolyP II

Value conversion (one direction):

$$\text{from}_r : \{A \ R : \text{Set}\} (C : \text{Code}_r) \rightarrow \llbracket C \rrbracket_r \ R \rightarrow \llbracket \uparrow^p C \rrbracket_p \ A \ R$$

$$\text{from}_r \ U_r = \text{id}$$

$$\text{from}_r \ I_r = \text{id}$$

$$\text{from}_r (F \oplus_r G) = [\text{inj}_1 \circ \text{from}_r F, \text{inj}_2 \circ \text{from}_r G]$$

$$\text{from}_r (F \otimes_r G) = \langle \text{from}_r F \circ \text{proj}_1, \text{from}_r G \circ \text{proj}_2 \rangle$$

$$\text{from}\mu_r : \{A : \text{Set}\} (C : \text{Code}_r) \rightarrow \mu_r C \rightarrow \mu_p (\uparrow^p C) \ A$$

$$\text{from}\mu_r C \langle x \rangle_r = \langle \text{from}_r C (\text{map}_r C (\text{from}\mu_r C) x) \rangle_p$$

Regular to PolyP III

Isomorphism proof (one direction):

$$\text{iso}_1 : \{A \ R : \text{Set}\} (C : \text{Code}_r) \{x : \llbracket C \rrbracket_r \ R\} \\ \rightarrow \text{to}_r \{A\} \ C \ (\text{from}_r \ C \ x) \equiv x$$

$$\text{iso}_1 \ U_r = \text{refl}$$

$$\text{iso}_1 \ I_r = \text{refl}$$

$$\text{iso}_1 (F \oplus_r G) \{ \text{inj}_1 \ _ \} = \text{cong} \ \text{inj}_1 \ (\text{iso}_1 \ F)$$

$$\text{iso}_1 (F \oplus_r G) \{ \text{inj}_2 \ _ \} = \text{cong} \ \text{inj}_2 \ (\text{iso}_1 \ G)$$

$$\text{iso}_1 (F \otimes_r G) \{ _ , _ \} = \text{cong}_2 \ _ , _ \ (\text{iso}_1 \ F) \ (\text{iso}_1 \ G)$$

Regular to PolyP IV

$\text{iso}\mu_1 : \{A : \text{Set}\} (C : \text{Code}_r) (x : \mu_r C)$
 $\rightarrow \text{to}\mu_r \{A\} C (\text{from}\mu_r C x) \equiv x$

$\text{iso}\mu_1 \{A\} C \langle x \rangle_r = \text{cong } \langle _ \rangle_r \$ \text{begin}$
 $\text{to}_r \{A\} C (\text{map}_p (\uparrow^p C) \text{id } (\text{to}\mu_r C)$
 $(\text{from}_r C (\text{map}_r C (\text{from}\mu_r C) x)))$

Regular to PolyP IV

$\text{iso}\mu_1 : \{A : \text{Set}\} (C : \text{Code}_r) (x : \mu_r C)$
 $\rightarrow \text{to}\mu_r \{A\} C (\text{from}\mu_r C x) \equiv x$

$\text{iso}\mu_1 \{A\} C \langle x \rangle_r = \text{cong } \langle _ \rangle_r \$ \text{begin}$

$\text{to}_r \{A\} C (\text{map}_p (\uparrow^p C) \text{id} (\text{to}\mu_r C)$
 $(\text{from}_r C (\text{map}_r C (\text{from}\mu_r C) x)))$

$\equiv \langle \text{mc } \{A\} C \rangle$

$\text{map}_r C (\text{to}\mu_r \{A\} C)$
 $(\text{to}_r \{A\} C (\text{from}_r C (\text{map}_r C (\text{from}\mu_r C) x)))$

Regular to PolyP IV

$$\text{iso}\mu_1 : \{A : \text{Set}\} (C : \text{Code}_r) (x : \mu_r C) \\ \rightarrow \text{to}\mu_r \{A\} C (\text{from}\mu_r C x) \equiv x$$

$$\text{iso}\mu_1 \{A\} C \langle x \rangle_r = \text{cong } \langle _ \rangle_r \$ \text{ begin} \\ \text{to}_r \{A\} C (\text{map}_p (\uparrow^P C) \text{id } (\text{to}\mu_r C) \\ (\text{from}_r C (\text{map}_r C (\text{from}\mu_r C) x))) \\ \equiv \langle \text{mc } \{A\} C \rangle \\ \text{map}_r C (\text{to}\mu_r \{A\} C) \\ (\text{to}_r \{A\} C (\text{from}_r C (\text{map}_r C (\text{from}\mu_r C) x)))$$

$$\text{mc} : \{A R_1 R_2 : \text{Set}\} \{x : _ \} \{f : R_1 \rightarrow R_2\} (C : \text{Code}_r) \\ \rightarrow \text{to}_r \{A\} \{R_2\} C (\text{map}_p (\uparrow^P C) \text{id } f x) \equiv \text{map}_r C f (\text{to}_r C x)$$

Regular to PolyP V

$$\text{map}_r C (\text{to}_{\mu_r} \{A\} C) \\ (\underline{\text{to}_r} \{A\} C (\underline{\text{from}_r} C (\text{map}_r C (\text{from}_{\mu_r} C) x)))$$

Regular to PolyP V

$$\begin{aligned} & \text{map}_r C (\text{to}_{\mu_r} \{A\} C) \\ & \quad (\underline{\text{to}_r} \{A\} C (\underline{\text{from}_r} C (\text{map}_r C (\text{from}_{\mu_r} C) x))) \\ \equiv & \langle \text{cong} (\text{map}_r C (\text{to}_{\mu_r} \{A\} C)) (\text{iso}_1 C) \rangle \\ & \quad \underline{\text{map}_r} C (\text{to}_{\mu_r} \{A\} C) (\underline{\text{map}_r} C (\text{from}_{\mu_r} \{A\} C) x) \end{aligned}$$

Regular to PolyP V

$$\begin{aligned} & \text{map}_r C (\text{to}_{\mu_r} \{A\} C) \\ & \quad (\underline{\text{to}}_r \{A\} C (\underline{\text{from}}_r C (\text{map}_r C (\text{from}_{\mu_r} C) x))) \\ \equiv & \langle \text{cong} (\text{map}_r C (\text{to}_{\mu_r} \{A\} C)) (\text{iso}_1 C) \rangle \\ & \quad \underline{\text{map}}_r C (\text{to}_{\mu_r} \{A\} C) (\underline{\text{map}}_r C (\text{from}_{\mu_r} \{A\} C) x) \\ \equiv & \langle \text{map}_r^\circ C \rangle \\ & \quad \text{map}_r C (\underline{\text{to}}_{\mu_r} C \circ \underline{\text{from}}_{\mu_r} C) x \end{aligned}$$

Regular to PolyP V

$$\begin{aligned} & \text{map}_r C (\text{to}_{\mu_r} \{A\} C) \\ & \quad (\text{to}_r \{A\} C (\text{from}_r C (\text{map}_r C (\text{from}_{\mu_r} C) x))) \\ \equiv & \langle \text{cong} (\text{map}_r C (\text{to}_{\mu_r} \{A\} C)) (\text{iso}_1 C) \rangle \\ & \quad \text{map}_r C (\text{to}_{\mu_r} \{A\} C) (\text{map}_r C (\text{from}_{\mu_r} \{A\} C) x) \\ \equiv & \langle \text{map}_r^\circ C \rangle \\ & \quad \text{map}_r C (\text{to}_{\mu_r} C \circ \text{from}_{\mu_r} C) x \\ \equiv & \langle \text{map}_r^\forall C (\text{to}_{\mu_r} C \circ \text{from}_{\mu_r} C) \text{id} (\text{iso}_{\mu_1} C) x \rangle \\ & \quad \text{map}_r C \text{id } x \\ \equiv & \langle \text{map}_r^{\text{id}} C \rangle \\ & \quad x \quad \square \end{aligned}$$

PolyP to InstantGenerics

$\text{p}\uparrow^{\text{ig}} : \text{Code}_p \rightarrow \text{Set} \rightarrow \text{Code}_{\text{ig}}$

$\text{p}\uparrow^{\text{ig}} C A = \text{p}\uparrow^{\text{ig}} \cdot C C A$ **where**

$\text{p}\uparrow^{\text{ig}} \cdot : \text{Code}_p \rightarrow \text{Code}_p \rightarrow \text{Set} \rightarrow \text{Code}_{\text{ig}}$

$\text{p}\uparrow^{\text{ig}} \cdot C U_p \quad A = U_{\text{ig}}$

$\text{p}\uparrow^{\text{ig}} \cdot C I_p \quad A = R_{\text{ig}} (\# \text{p}\uparrow^{\text{ig}} \cdot C C A)$

$\text{p}\uparrow^{\text{ig}} \cdot C P_p \quad A = K_{\text{ig}} A$

$\text{p}\uparrow^{\text{ig}} \cdot C (F \oplus_p G) A = (\text{p}\uparrow^{\text{ig}} \cdot C F A) \oplus_{\text{ig}} (\text{p}\uparrow^{\text{ig}} \cdot C G A)$

$\text{p}\uparrow^{\text{ig}} \cdot C (F \otimes_p G) A = (\text{p}\uparrow^{\text{ig}} \cdot C F A) \otimes_{\text{ig}} (\text{p}\uparrow^{\text{ig}} \cdot C G A)$

$\text{p}\uparrow^{\text{ig}} \cdot C (F \odot_p G) A = R_{\text{ig}} (\# \text{p}\uparrow^{\text{ig}} \cdot F F \llbracket (\text{p}\uparrow^{\text{ig}} \cdot C G A) \rrbracket_{\text{ig}})$

Indexed to InstantGenerics

$$i\uparrow^{\text{ig}} : \{ | O : \text{Set} \} \\ \rightarrow \text{Code}_i | O \rightarrow (I \rightarrow \text{Set}) \rightarrow (O \rightarrow \text{Code}_{\text{ig}})$$

$$i\uparrow^{\text{ig}} C r o = i\uparrow^{\text{ig}} \cdot C (K_{\text{ig}} \circ r) \circ \mathbf{where}$$

$$i\uparrow^{\text{ig}} \cdot : \{ | O : \text{Set} \} \\ \rightarrow \text{Code}_i | O \rightarrow (I \rightarrow \text{Code}_{\text{ig}}) \rightarrow (O \rightarrow \text{Code}_{\text{ig}})$$

$$i\uparrow^{\text{ig}} \cdot U_i \quad r o = U_{\text{ig}}$$

$$i\uparrow^{\text{ig}} \cdot (|_i i) \quad r o = r i$$

$$i\uparrow^{\text{ig}} \cdot (!_i i) \quad r o = K_{\text{ig}} (o \equiv i)$$

$$i\uparrow^{\text{ig}} \cdot (F \oplus_i G) r o = (i\uparrow^{\text{ig}} \cdot F r o) \oplus_{\text{ig}} (i\uparrow^{\text{ig}} \cdot G r o)$$

$$i\uparrow^{\text{ig}} \cdot (F \otimes_i G) r o = (i\uparrow^{\text{ig}} \cdot F r o) \otimes_{\text{ig}} (i\uparrow^{\text{ig}} \cdot G r o)$$

$$i\uparrow^{\text{ig}} \cdot (F \odot_i G) r o = R_{\text{ig}} (\# i\uparrow^{\text{ig}} \cdot F (i\uparrow^{\text{ig}} \cdot G r) o)$$

$$i\uparrow^{\text{ig}} \cdot (\text{Fix}_i F) r o = R_{\text{ig}} (\# i\uparrow^{\text{ig}} \cdot F [r, i\uparrow^{\text{ig}} \cdot (\text{Fix}_i F) r] o)$$

Outline

Introduction

Generic programming libraries, in Agda

Regular

PolyP

Indexed functors

Instant generics

Conversions

Conclusion

Conclusion

- ▶ Analyzed five libraries, with an encoding in Agda for each
- ▶ Relations between the fixed-point approaches made (formally) clear
- ▶ The same generic function can be used in different libraries
- ▶ Embedding into InstantGenerics requires expanding fixed points, highlighting e.g. the way composition works in PolyP

Conclusion

- ▶ Analyzed five libraries, with an encoding in Agda for each
- ▶ Relations between the fixed-point approaches made (formally) clear
- ▶ The same generic function can be used in different libraries
- ▶ Embedding into InstantGenerics requires expanding fixed points, highlighting e.g. the way composition works in PolyP

Future work:

- ▶ Prove termination, remove `--type-in-type`
- ▶ Expand to other generic views