Syntax-Generic Operations, Reflectively Reified

Extended Abstract

Tzu-Chi Lin and Josh Ko

September 9, 2022

Institute of Information Science
Academia Sinica, Taiwan

Typical Languages in Languages

Intrinsic typing is common for λ-calculus with De Bruijn indices.

$$
\begin{aligned}
&\textbf{data } \_\vdash\_ \;:\; \mathit{Context} \to \mathit{Ty} \to \mathit{Set} \textbf{ where}\\
&\quad `\_ \;:\; \Gamma \ni A \to \Gamma \vdash A\\
&\quad \lambda\_ \;:\; \Gamma , A \vdash B \to \Gamma \vdash A \Rightarrow B\\
&\quad \_\cdot\_ \;:\; \Gamma \vdash A \Rightarrow B \to \Gamma \vdash A \to \Gamma \vdash B
\end{aligned}
$$

$$
\begin{aligned}
&\textbf{data } \mathit{Ty} \;:\; \mathit{Set} \textbf{ where}\\
&\quad \alpha \;:\; \mathit{Ty}\\
&\quad \_\Rightarrow\_ \;:\; \mathit{Ty} \to \mathit{Ty} \to \mathit{Ty}\\
&\textbf{data } \mathit{Context}\ (A \;:\; \mathit{Set}) \;:\; \mathit{Set} \textbf{ where}\\
&\quad \emptyset \;:\; \mathit{Context}\ A\\
&\quad \_,\_ \;:\; (\Gamma \;:\; \mathit{Context}\ A) \to A \to \mathit{Context}\ A\\
&\textbf{data } \_\ni\_ \;:\; \mathit{Context} \to \mathit{Ty} \to \mathit{Set} \textbf{ where}\\
&\quad z \;:\; \Gamma , A \ni A\\
&\quad s \;:\; \Gamma \ni A \to \Gamma , B \ni A
\end{aligned}
$$

Scope-safe syntax operations:

$$
\begin{aligned}
&\mathit{rename} : \forall\ \{\Gamma\ \Delta\} \to (\forall\ \{A\} \to \Gamma \ni A \to \Delta \ni A) \\
&\qquad\qquad\qquad\quad \to (\forall\ \{A\} \to \Gamma \vdash A \to \Delta \vdash A) \\
&\mathit{rename}\ \rho\ ({}^{\backprime}\ x) \quad = {}^{\backprime}\ (\rho\ x) \\
&\mathit{rename}\ \rho\ (\lambda\!\!\!-\ N) \quad = \lambda\!\!\!-\ (\mathit{rename}\ (\mathit{ext}\ \rho)\ N) \\
&\mathit{rename}\ \rho\ (L \cdot M) \quad = (\mathit{rename}\ \rho\ L) \cdot (\mathit{rename}\ \rho\ M)
\end{aligned}
$$

Motivation

Intrinsic typing is common for λ-calculus with De Bruijn indices.

$$
\begin{array}{l}
\textbf{data}\ \mathit{Ty}\ :\ \mathit{Set}\ \textbf{where} \\
\quad \ldots \\
\quad \text{`}\mathbb{N}\ :\ \mathit{Ty} \\
\textbf{data}\ \_{\vdash}\_\ :\ \mathit{Context} \to \mathit{Ty} \to \mathit{Set}\ \textbf{where} \\
\quad \ldots \\
\quad \begin{array}{lll}
\text{`}\mathit{zero} & : & \Gamma \vdash \text{`}\mathbb{N} \\
\text{`}\mathit{suc}\_ & : & \Gamma \vdash \text{`}\mathbb{N} \to \Gamma \vdash \text{`}\mathbb{N} \\
\mathit{case} & : & \Gamma \vdash \text{`}\mathbb{N} \to \Gamma \vdash A \to \Gamma\ ,\ \text{`}\mathbb{N} \vdash A \to \Gamma \vdash A \\
\mu\_ & : & \Gamma\ ,\ A \vdash A \to \Gamma \vdash A
\end{array}
\end{array}
$$

Motivation

$$
\begin{aligned}
&\mathit{rename} : \forall\, \{\Gamma\ \Delta\} \to (\forall\, \{A\} \to \Gamma \ni A \to \Delta \ni A) \\
&\qquad\qquad\qquad\quad \to (\forall\, \{A\} \to \Gamma \vdash A \to \Delta \vdash A) \\
&\ldots \\
&\mathit{rename}\ \rho\ (\text{`}\mathit{zero}) \qquad\quad\ = \text{`}\mathit{zero} \\
&\mathit{rename}\ \rho\ (\text{`}\mathit{suc}\ M) \qquad\ = \text{`}\mathit{suc}\ (\mathit{rename}\ \rho\ M) \\
&\mathit{rename}\ \rho\ (\mathit{case}\ L\ M\ N) = \mathit{case}\ (\mathit{rename}\ \rho\ L) \\
&\qquad\qquad\qquad\qquad\qquad\qquad (\mathit{rename}\ \rho\ M) \\
&\qquad\qquad\qquad\qquad\qquad\qquad (\mathit{rename}\ (\mathit{ext}\ \rho)\ N) \\
&\mathit{rename}\ \rho\ (\mu\ N) \qquad\qquad = \mu\ (\mathit{rename}\ (\mathit{ext}\ \rho)\ N)
\end{aligned}
$$

Motivation

Change/extend the object language even further:

$$
\begin{array}{lll}
\textbf{data}\ \_\vdash\_ & : & \mathit{Context} \to \mathit{Ty} \to \mathit{Set}\ \textbf{where} \\
\quad \ldots & & \\
\quad \mathit{con} & : & \mathbb{N} \to \Gamma \vdash \mathit{Nat} \\
\quad \_`{*}\_ & : & \Gamma \vdash \mathit{Nat} \to \Gamma \vdash \mathit{Nat} \to \Gamma \vdash \mathit{Nat} \\
\quad `\mathit{let} & : & \Gamma \vdash A \to \Gamma\ ,\ A \vdash B \to \Gamma \vdash B \\
\quad `\!<\_\ ,\ \_\!> & : & \Gamma \vdash A \to \Gamma \vdash B \to \Gamma \vdash A\ `{\times}\ B \\
\quad `\mathit{proj}_1 & : & \Gamma \vdash A\ `{\times}\ B \to \Gamma \vdash A \\
\quad `\mathit{proj}_2 & : & \Gamma \vdash A\ `{\times}\ B \to \Gamma \vdash B \\
\quad \mathit{case}{\times} & : & \Gamma \vdash A\ `{\times}\ B \to \Gamma\ ,\ A\ ,\ B \vdash C \to \Gamma \vdash C
\end{array}
$$

Motivation

Redefine/extend syntax operations:

$$
\begin{array}{lcl}
\ldots & & \\
\mathit{rename}\ \rho\ (\mathit{con}\ n) & = & \mathit{con}\ n \\
\mathit{rename}\ \rho\ (M\ \grave{}{*}\ N) & = & \mathit{rename}\ \rho\ M\ \grave{}{*}\ \mathit{rename}\ \rho\ N \\
\mathit{rename}\ \rho\ (\grave{}\mathit{let}\ M\ N) & = & \grave{}\mathit{let}\ (\mathit{rename}\ \rho\ M)\ (\mathit{rename}\ (\mathit{ext}\ \rho)\ N) \\
\mathit{rename}\ \rho\ \grave{}{<}\ M\ ,\ N\ {>} & = & \grave{}{<}\ \mathit{rename}\ \rho\ M\ ,\ \mathit{rename}\ \rho\ N\ {>} \\
\mathit{rename}\ \rho\ (\grave{}\mathit{proj}_1\ L) & = & \grave{}\mathit{proj}_1\ (\mathit{rename}\ \rho\ L) \\
\mathit{rename}\ \rho\ (\grave{}\mathit{proj}_2\ L) & = & \grave{}\mathit{proj}_2\ (\mathit{rename}\ \rho\ L) \\
\mathit{rename}\ \rho\ (\mathit{case}{\times}\ L\ M) & = & \mathit{case}{\times}\ (\mathit{rename}\ \rho\ L) \\
 & & \qquad (\mathit{rename}\ (\mathit{ext}\ (\mathit{ext}\ \rho))\ M)
\end{array}
$$

Motivation

Other repeating operations:

$$
\begin{aligned}
\mathit{subst} &: \forall\ \{\Gamma\ \Delta\} \to (\forall\ \{A\} \to \Gamma \ni A \to \Delta \vdash A) \\
&\qquad\qquad\quad \to (\forall\ \{A\} \to \Gamma \vdash A \to \Delta \vdash A) \\
\mathit{print} &: \Gamma \vdash A \to \mathit{String} \\
&\ldots
\end{aligned}
$$

Where we are going...

1. Existing Work for Syntax-generic Operations
2. Elaborator Reflection to the Rescue
3. Discussion

Existing Work for Syntax-generic Operations

Existing work

There are generic libraries for a family/families of syntaxes with binders.

We improve upon Allais et al.'s approach presented at ICFP '18 (later published in JFP '21).

Existing work by Allais et al.

Allais et al.'s *Desc*:

$$
\begin{array}{l}
\textbf{data}\ \mathit{Desc}\ (I : \mathit{Set}) : \mathit{Set}_1\ \textbf{where} \\
\quad \text{`}\sigma : (A : \mathit{Set}) \to (A \to \mathit{Desc}\ I) \to \mathit{Desc}\ I \\
\quad \text{`}X : \mathit{List}\ I \to I \to \mathit{Desc}\ I \to \mathit{Desc}\ I \\
\quad \text{`}\blacksquare : I \to \mathit{Desc}\ I
\end{array}
$$

$$
\begin{array}{l}
\textbf{data}\ \text{`}\mathit{STLC} : \mathit{Set}\ \textbf{where} \\
\quad \mathit{App}\ \mathit{Lam} : \mathit{Ty} \to \mathit{Ty} \to \text{`}\mathit{STLC} \\
\mathit{STLCD} : \mathit{Desc}\ \mathit{Ty} \\
\mathit{STLCD} = \text{`}\sigma\ \text{`}\mathit{STLC}\ \lambda\ \textbf{where} \\
\quad (\mathit{App}\ i\ j) \to \\
\qquad \text{`X}\ [\,]\ (i \Rightarrow j)\ (\text{`X}\ [\,]\ i\ (\blacksquare\ j)) \\
\quad (\mathit{Lam}\ i\ j) \to \\
\qquad \text{`X}\ (i :: [\,])\ j\ (\blacksquare\ (i \Rightarrow j))
\end{array}
$$

Existing work by Allais et al.

Simply typed λ-calculus

$$
\begin{aligned}
&\textbf{data}\ \_\vdash\_\ :\ \mathit{Context} \to \mathit{Ty} \to \mathit{Set}\ \textbf{where} \\
&\quad \text{`}\_\ \ :\ \Gamma \ni A \to \Gamma \vdash A \\
&\quad \lambda\_\ :\ \Gamma\ ,\ A \vdash B \to \Gamma \vdash A \Rightarrow B \\
&\quad \_\cdot\_\ :\ \Gamma \vdash A \Rightarrow B \to \Gamma \vdash A \to \Gamma \vdash B
\end{aligned}
$$

Existing work by Allais et al.

Simply typed λ-calculus

$$\begin{array}{l}
\mathbf{data}\ _\vdash_\ :\ \mathit{Context} \to \mathit{Ty} \to \mathit{Set}\ \mathbf{where} \\
\quad \text{`}_\ :\ \Gamma \ni A \to \Gamma \vdash A \\
\quad \lambda_\ :\ \Gamma\ ,\ A \vdash B \to \Gamma \vdash A \Rightarrow B \\
\quad _\cdot_\ :\ \Gamma \vdash A \Rightarrow B \to \Gamma \vdash A \to \Gamma \vdash B
\end{array}$$

encoded in *Desc*:

$$\begin{array}{l}
\mathit{STLCD}\ :\ \mathit{Desc}\ \mathit{Ty} \\
\mathit{STLCD}\ =\ \text{`}\sigma\ \text{`}\mathit{STLC}\ \lambda\ \mathbf{where} \\
\quad (\mathit{App}\ i\ j) \to \\
\qquad \text{`X}\ []\ (i \Rightarrow j)\ (\text{`X}\ []\ i\ (\blacksquare\ j)) \\
\quad (\mathit{Lam}\ i\ j) \to \\
\qquad \text{`X}\ (i :: [])\ j\ (\blacksquare\ (i \Rightarrow j))
\end{array}$$

Existing work by Allais et al.

Simply typed λ-calculus

$$
\begin{aligned}
&\textbf{data}\ \_{\vdash}\_ \ :\ \mathit{Context} \to \mathit{Ty} \to \mathit{Set}\ \textbf{where}\\
&\quad \text{`}\_ \ :\ \Gamma \ni A \to \Gamma \vdash A\\
&\quad \lambda\_ \ :\ \Gamma\ ,\ A \vdash B \to \Gamma \vdash A \Rightarrow B\\
&\quad \_\cdot\_ \ :\ \Gamma \vdash A \Rightarrow B \to \Gamma \vdash A \to \Gamma \vdash B
\end{aligned}
$$

encoded in *Desc*:

$$
\begin{aligned}
&\mathit{STLCD}\ :\ \mathit{Desc}\ \mathit{Ty}\\
&\mathit{STLCD}\ =\ \text{`}\sigma\ \text{`}\mathit{STLC}\ \lambda\ \textbf{where}\\
&\quad (\mathit{App}\ i\ j) \to\\
&\qquad \text{`X}\ [\,]\ (i \Rightarrow j)\ (\text{`X}\ [\,]\ i\ (\blacksquare\ j))\\
&\quad (\mathit{Lam}\ i\ j) \to\\
&\qquad \text{`X}\ (i :: [\,])\ j\ (\blacksquare\ (i \Rightarrow j))
\end{aligned}
$$

$$\mathit{STLC}'\ =\ \mathit{Tm}\ \mathit{STLCD}$$

Existing work by Allais et al.

Simply typed λ-calculus

$$
\begin{aligned}
&\textbf{data}\ \_\vdash\_ \ :\ \mathit{Context} \to \mathit{Ty} \to \mathit{Set}\ \textbf{where}\\
&\quad \text{`}\_ \ :\ \Gamma \ni A \to \Gamma \vdash A\\
&\quad \lambda\_ \ :\ \Gamma\ , A \vdash B \to \Gamma \vdash A \Rightarrow B\\
&\quad \_\cdot\_ \ :\ \Gamma \vdash A \Rightarrow B \to \Gamma \vdash A \to \Gamma \vdash B
\end{aligned}
$$

encoded in *Desc*:

$$
\begin{aligned}
&\mathit{STLCD}\ :\ \mathit{Desc}\ \mathit{Ty}\\
&\mathit{STLCD}\ =\ \text{`}\sigma\ \text{`}\mathit{STLC}\ \lambda\ \textbf{where}\\
&\quad (\mathit{App}\ i\ j) \to\\
&\qquad \text{`}X\ []\ (i \Rightarrow j)\ (\text{`}X\ []\ i\ (\blacksquare\ j))\\
&\quad (\mathit{Lam}\ i\ j) \to\\
&\qquad \text{`}X\ (i :: [])\ j\ (\blacksquare\ (i \Rightarrow j))
\end{aligned}
$$

$$
\mathit{STLC}' \ =\ \mathit{Tm}\ \mathit{STLCD}
$$

$$
\_\vdash\_ \cong \mathit{STLC}'
$$

Generic Functions for the Whole Universe

$$
\begin{array}{l}
\mathit{Var}\ \sigma\ \Gamma\ \ =\ \ \Gamma \ni \sigma \\
\\
\mathit{Renaming}\ :\ \forall\ \{d\ :\ \mathit{Desc}\ I\} \to \mathit{Semantics}\ d\ \mathit{Var}\ (\mathit{Tm}\ d) \\
\\
\mathit{rename}\ :\ \forall\ \{d\ :\ \mathit{Desc}\ I\} \to (\forall\ \{\sigma\} \to \mathit{Var}\ \sigma\ \Gamma\ \to \mathit{Var}\ \sigma\ \Delta) \\
\quad \forall\ \{\tau\} \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \to \mathit{Tm}\ d\ \tau\ \Gamma \to \mathit{Tm}\ d\ \tau\ \Delta \\
\mathit{rename}\ \ =\ \ \mathit{semantics}\ \mathit{Renaming}
\end{array}
$$

Generic Functions for the Whole Universe

$$Var\ \sigma\ \Gamma\ =\ \Gamma \ni \sigma$$

$$Renaming\ :\ \forall\ \{d\ :\ Desc\ I\} \to Semantics\ d\ Var\ (Tm\ d)$$

$$\begin{aligned}
&rename\ :\ \forall\ \{d\ :\ Desc\ I\} \to (\forall\ \{\sigma\} \to Var\ \sigma\ \Gamma \to Var\ \sigma\ \Delta) \\
&\quad \forall\ \{\tau\} \qquad\qquad\qquad\qquad\qquad\qquad \to Tm\ d\ \tau\ \Gamma \to Tm\ d\ \tau\ \Delta \\
&rename\ =\ semantics\ Renaming
\end{aligned}$$

Generic programs are *Semantics* records.

Generic Functions for the Whole Universe

$$
\begin{aligned}
&\mathit{Var}\ \sigma\ \Gamma \ = \ \Gamma \ni \sigma \\
&\mathit{Renaming} : \forall \{d : \mathit{Desc}\ I\} \to \mathit{Semantics}\ d\ \mathit{Var}\ (\mathit{Tm}\ d) \\
&\mathit{rename} : \forall \{d : \mathit{Desc}\ I\} \to (\forall \{\sigma\} \to \mathit{Var}\ \sigma\ \Gamma \ \to \mathit{Var}\ \sigma\ \Delta) \\
&\quad \forall \{\tau\} \qquad\qquad\qquad\qquad\qquad\qquad \to \mathit{Tm}\ d\ \tau\ \Gamma \to \mathit{Tm}\ d\ \tau\ \Delta \\
&\mathit{rename} \ = \ \mathit{semantics}\ \mathit{Renaming}
\end{aligned}
$$

Generic programs are *Semantics* records.

Functions are realized on fixpoints *Tm* via *semantics*.

Generic Functions for the Whole Universe

$$
\begin{aligned}
&\mathit{Var}\ \sigma\ \Gamma \;=\; \Gamma \ni \sigma \\
&\mathit{Renaming} : \forall \{d : \mathit{Desc}\ I\} \to \mathit{Semantics}\ d\ \mathit{Var}\ (\mathit{Tm}\ d) \\
&\mathit{rename} : \forall \{d : \mathit{Desc}\ I\} \to (\forall \{\sigma\} \to \mathit{Var}\ \sigma\ \Gamma \to \mathit{Var}\ \sigma\ \Delta) \\
&\quad \forall \{\tau\} \qquad\qquad\qquad\qquad\qquad\qquad\quad \to \mathit{Tm}\ d\ \tau\ \Gamma \to \mathit{Tm}\ d\ \tau\ \Delta \\
&\mathit{rename} = \mathit{semantics}\ \mathit{Renaming}
\end{aligned}
$$

Generic programs are *Semantics* records.

Functions are realized on fixpoints *Tm* via *semantics*.

rename can be applied to fixpoints of any description (e.g. *Tm STLCD*).

Problems with Syntax Universes: Readability

$$
\begin{array}{lcl}
\textbf{data}\ \_\vdash\_ & : & \mathit{Context} \to \mathit{Ty} \to \mathit{Set}\ \textbf{where} \\
\quad `\_ & : & \Gamma \ni A \to \Gamma \vdash A \\
\quad ƛ\_ & : & \Gamma , A \vdash B \to \Gamma \vdash A \Rightarrow B \\
\quad \_\cdot\_ & : & \Gamma \vdash A \Rightarrow B \to \Gamma \vdash A \to \Gamma \vdash B
\end{array}
$$

$$
\begin{array}{l}
\mathit{STLCD}\ :\ \mathit{Desc}\ \mathit{Ty} \\
\mathit{STLCD}\ =\ `\sigma\ `\mathit{STLC}\ \lambda\ \textbf{where} \\
\quad (\mathit{App}\ i\ j) \to \\
\qquad `\mathrm{X}\ []\ (i \Rightarrow j)\ (`\mathrm{X}\ []\ i\ (\blacksquare\ j)) \\
\quad (\mathit{Lam}\ i\ j) \to \\
\qquad `\mathrm{X}\ (i :: [])\ j\ (\blacksquare\ (i \Rightarrow j)) \\
\mathit{STLC}'\ =\ \mathit{Tm}\ \mathit{STLCD}
\end{array}
$$

Problems with Syntax Universes: Burden on Programmers

$$
\begin{array}{l}
\textbf{data}\ \mathit{Desc}\ (I\ :\ \mathit{Set})\ :\ \mathit{Set}_1\ \textbf{where} \\
\quad \text{`}\sigma\ :\ (A\ :\ \mathit{Set}) \to (A \to \mathit{Desc}\ I) \to \mathit{Desc}\ I \\
\quad \text{`}X\ :\ \mathit{List}\ I \to I \to \mathit{Desc}\ I \to \mathit{Desc}\ I \\
\quad \text{`}\blacksquare\ :\ I \to \mathit{Desc}\ I
\end{array}
$$

Problems with Syntax Universes: Interoperability

$$
\begin{aligned}
STLCD &: Desc\ Ty \\
STLCD &= \ldots
\end{aligned}
$$

$$
\begin{aligned}
STLCD' &: Desc'\ ??? \\
STLCD' &= \ ???
\end{aligned}
$$

Problems with Syntax Universes: Interoperability

$$
\begin{aligned}
f &: \Gamma \vdash A \to \mathbb{N} \\
f\ t &= \{!\ \mid\ !\}
\end{aligned}
$$

Ctrl + C $\downarrow$

Problems with Syntax Universes: Interoperability

$$
\begin{aligned}
f &: \Gamma \vdash A \to \mathbb{N} \\
f\ t &= \{!\ \mid\ !\}
\end{aligned}
$$

Ctrl + C ↓

$$
\begin{aligned}
&f : \Gamma \vdash A \to \mathbb{N} \\
&f\ (\text{`}\ x) && = \{!\quad !\} \\
&f\ (\lambda\ N) && = \{!\quad !\} \\
&f\ (L \cdot M) && = \{!\ \mid\ !\}
\end{aligned}
$$

Problems with Syntax Universes: Interoperability

$$f : \Gamma \vdash A \to \mathbb{N}$$
$$f\ t = \{!\ |\ !\}$$

Ctrl + C ↓

$$\begin{aligned} f &: \Gamma \vdash A \to \mathbb{N} \\ f\ (` x) &= \{!\quad !\} \\ f\ (ƛ\ N) &= \{!\quad !\} \\ f\ (L \cdot M) &= \{!\ |\ !\} \end{aligned}$$

$$f : Tm\ d\ \tau\ \Gamma \to \mathbb{N}$$
$$f\ t = \{!\ |\ !\}$$

Ctrl + C ↓

???

Elaborator Reflection to the Rescue

Elaborator Reflection to the Rescue

“Datatype-Generic Programming Meets Elaborator Reflection” by Josh Ko, Liang-Ting Chen, and Tzu-Chi Lin at 15:50, Tuesday.

Elaborator Reflection to the Rescue

"Datatype-Generic Programming Meets Elaborator Reflection" by Josh Ko, Liang-Ting Chen, and Tzu-Chi Lin at 15:50, Tuesday.

Syntax-generic operations *are* Datatype-generic programs with constraints.

1. The programmer defines a native datatype T.

The process

1. The programmer defines a native datatype T.
2. A metaprogram generates the description D of T.

The process

1. The programmer defines a native datatype T.
2. A metaprogram generates the description D of T.
3. The programmer chooses a description P from a set of pre-defined generic programs.

The process

1. The programmer defines a native datatype T.
2. A metaprogram generates the description D of T.
3. The programmer chooses a description P from a set of pre-defined generic programs.
4. A metaprogram takes D and P, generates a native function accordingly.

The process

1. The programmer defines a native datatype T.
2. A metaprogram generates the description D of T.
3. The programmer provides a proof S of D that says T is indeed a syntax.
4. The programmer chooses a description P from a set of pre-defined generic programs.
5. A metaprogram takes D, S, and P, generates a native function accordingly.

Flow Chart

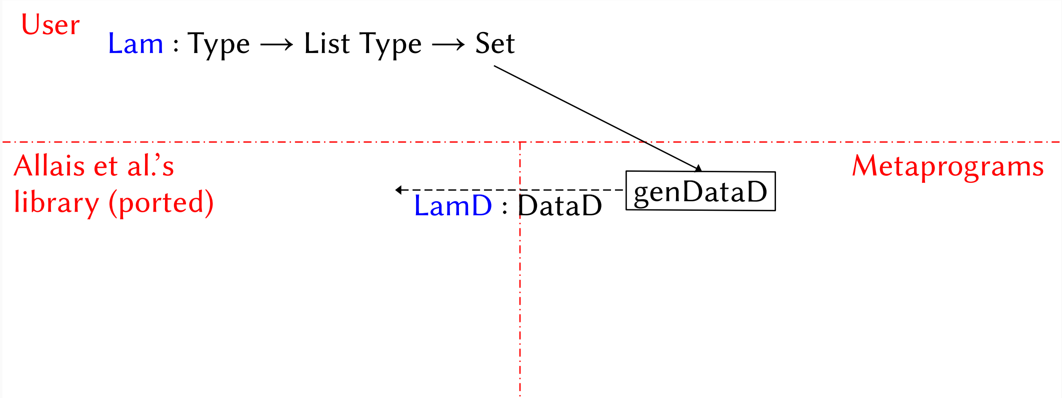
User

$\textcolor{blue}{\mathsf{Lam}} : \mathsf{Type} \longrightarrow \mathsf{List\ Type} \longrightarrow \mathsf{Set}$

Allais et al.'s library (ported)

Metaprograms

Flow Chart


User
Lam : Type ⟶ List Type ⟶ Set
Allais et al.'s
library (ported)
Metaprograms
genDataD
LamD : DataD

Flow Chart

User

Lam : Type ⟶ List Type ⟶ Set

LamSyn : Syntax LamD

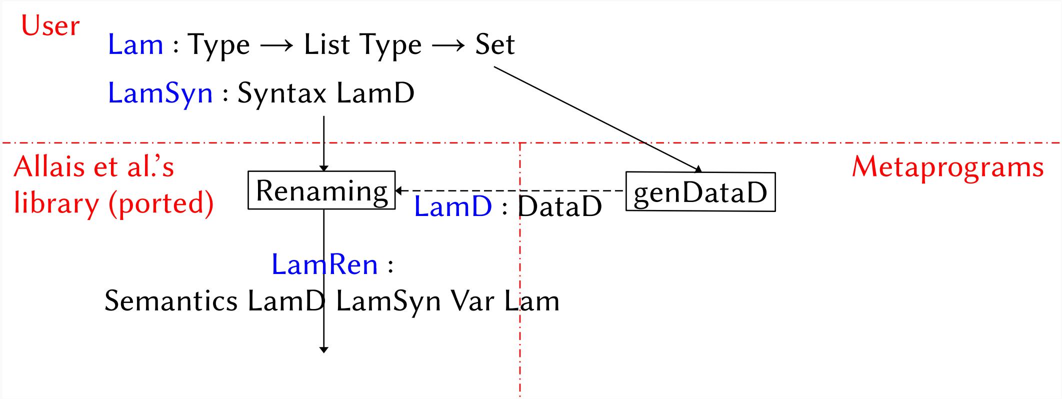
Allais et al.'s library (ported)

Metaprograms

$\boxed{\text{genDataD}}$

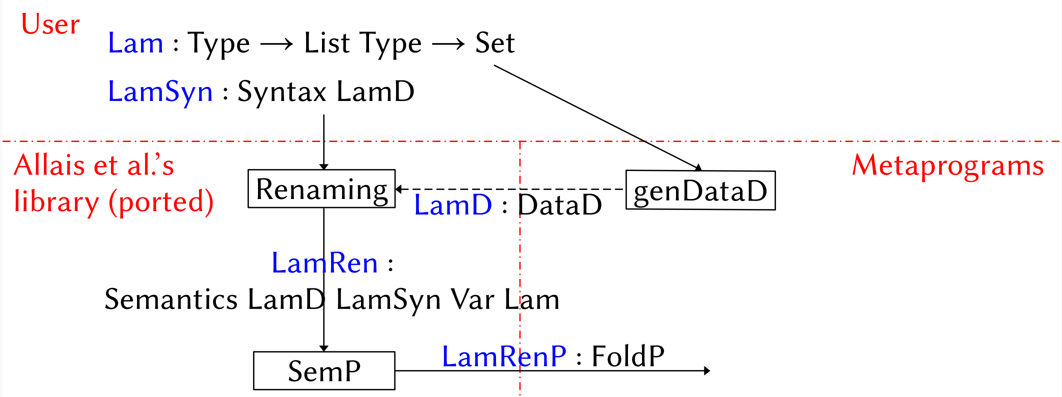
LamD : DataD

Flow Chart

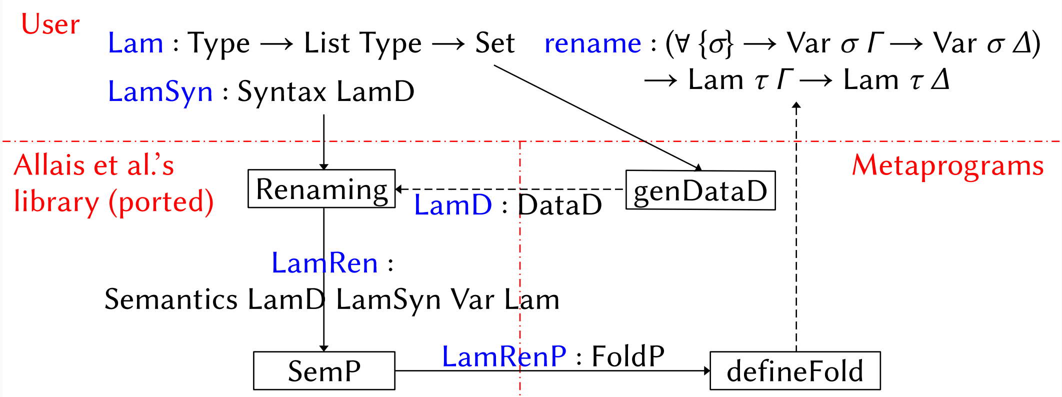


User
Lam : Type ⟶ List Type ⟶ Set
LamSyn : Syntax LamD
Allais et al.'s
library (ported)
Renaming
LamD : DataD
genDataD
Metaprograms
LamRen :
Semantics LamD LamSyn Var Lam

Flow Chart


User
Lam : Type ⟶ List Type ⟶ Set
LamSyn : Syntax LamD
Allais et al.'s
library (ported)
Renaming
LamD : DataD
genDataD
Metaprograms
LamRen :
Semantics LamD LamSyn Var Lam
SemP
LamRenP : FoldP

Flow Chart


User
Lam : Type → List Type → Set
LamSyn : Syntax LamD
rename : (∀ {σ} → Var σ Γ → Var σ Δ)
→ Lam τ Γ → Lam τ Δ
Allais et al.'s
library (ported)
Renaming
LamD : DataD
genDataD
Metaprograms
LamRen :
Semantics LamD LamSyn Var Lam
SemP
LamRenP : FoldP
defineFold

Some of Our Contributions

In “Datatype-generic Programming Meets Elaborator Reflection”:

- *DataD*
- *FoldP* for folds (and *IndP* for inductions)
- metaprogram *genDataD*
- metaprogram *defineFold*

Some of Our Contributions

In "Datatype-generic Programming Meets Elaborator Reflection":

- *DataD*
- *FoldP* for folds (and *IndP* for inductions)
- metaprogram *genDataD*
- metaprogram *defineFold*

In this work:

- predicate *Syntax* on *DataD* that captures *Desc*.
- function *SemP* that generates *FoldP* from *Syntax* proofs.

The *Syntax* Predicate

$$Syntax\ :\ Set\ \ell \rightarrow DataD \rightarrow Set\omega$$

Desc are captured by *Syntax* as each:

THE *Syntax* PREDICATE

$$Syntax \ : \ Set\ \ell \to DataD \to Set\omega$$

Desc are captured by *Syntax* as each:

- has a variable rule,

The *Syntax* Predicate

$$Syntax \;:\; Set\ \ell \to DataD \to Set\omega$$

Desc are captured by *Syntax* as each:

- has a variable rule,
- is not universe polymorphic,

The *Syntax* Predicate

$$Syntax \ : \ Set\ \ell \to DataD \to Set\omega$$

Desc are captured by *Syntax* as each:

- has a variable rule,
- is not universe polymorphic,
- has two indices, I and $List\ I$, and

The *Syntax* Predicate

$$\mathit{Syntax} \; : \; \mathit{Set}\;\ell \to \mathit{DataD} \to \mathit{Set}\omega$$

Desc are captured by *Syntax* as each:

- has a variable rule,
- is not universe polymorphic,
- has two indices, I and $\mathit{List}\ I$, and
- supports context extensions.

The *Syntax* Predicate

Does *PCF* satisfies *Syntax*?

$$
\begin{aligned}
&\textbf{data}\ \mathit{PCF}\ :\ \mathit{Ty} \to \mathit{Context} \to \mathit{Set}\ \textbf{where}\\
&\quad \text{`}\mathit{var}\ \ :\ \mathit{Var}\ \sigma\ \Gamma \to \mathit{PCF}\ \sigma\ \Gamma\\
&\quad \text{`}\mathit{app}\ \ :\ \mathit{PCF}\ (\sigma \Rightarrow \tau)\ \Gamma \to \mathit{PCF}\ \sigma\ \Gamma \to \mathit{PCF}\ \tau\ \Gamma\\
&\quad \text{`}\mathit{lam}\ \ :\ \mathit{PCF}\ \tau\ (\sigma :: \Gamma) \to \mathit{PCF}\ (\sigma \Rightarrow \tau)\ \Gamma\\
&\quad \text{`}\mathit{zero}\ :\ \mathit{PCF}\ \text{`}\mathbb{N}\ \Gamma\\
&\quad \text{`}\mathit{suc}\_\ :\ \mathit{PCF}\ \text{`}\mathbb{N}\ \Gamma \to \mathit{PCF}\ \text{`}\mathbb{N}\ \Gamma
\end{aligned}
$$

The *Syntax* Predicate

Does PCF satisfies $Syntax$?

$$
\begin{aligned}
&\textbf{data}\ PCF\ :\ Ty \to Context \to Set\ \textbf{where} \\
&\quad \grave{}var\ \ :\ Var\ \sigma\ \Gamma \to PCF\ \sigma\ \Gamma \\
&\quad \grave{}app\ \ :\ PCF\ (\sigma \Rightarrow \tau)\ \Gamma \to PCF\ \sigma\ \Gamma \to PCF\ \tau\ \Gamma \\
&\quad \grave{}lam\ \ :\ PCF\ \tau\ (\sigma :: \Gamma) \to PCF\ (\sigma \Rightarrow \tau)\ \Gamma \\
&\quad \grave{}zero\ :\ PCF\ \grave{}\mathbb{N}\ \Gamma \\
&\quad \grave{}suc\_\ :\ PCF\ \grave{}\mathbb{N}\ \Gamma \to PCF\ \grave{}\mathbb{N}\ \Gamma
\end{aligned}
$$

Proof of PCF being Syntax:

$$
\begin{aligned}
&SyntaxPCF\ :\ Syntax\ Ty\ (genDataD\ PCF) \\
&SyntaxPCF\ =\ \_ \\
&\quad ,\ refl \\
&\quad ,\ (refl\ , refl) \\
&\quad ,\ \_ \\
&\quad ,\ refl \\
&\quad ,\ refl \\
&\quad ,\ (\_\ ,\ \_\ ,\ \_\ ,\ refl\ ,\ (\lambda\ \_ \to refl)) \\
&\quad ,\ (\_\ ,\ \_\ ,\ \_\ ,\ refl\ ,\ (\lambda\ \_ \to refl)) \\
&\quad ,\ (\_\ ,\ \_\ ,\ \_\ ,\ refl\ ,\ (\lambda\ \_ \to refl)) \\
&\quad ,\ (\_\ ,\ \_\ ,\ \_\ ,\ refl\ ,\ (\lambda\ \_ \to refl)) \\
&\quad ,\ tt
\end{aligned}
$$

Discussion

Towards Datatype-generic Libraries for Syntaxes?

Do we really need syntax-generic libraries?