Type Theory in Type Theory using Quotient Inductive Types

Jakub Dreżewski, Tomasz Miśkowicz

19.12.2024

Motivation

- No preterms and typability relations $\Rightarrow$ directly well-typed objects defined inductively

Motivation

- No preterms and typability relations $\Rightarrow$ directly well-typed objects defined inductively
- No type preservation theorems needed $\Rightarrow$ preservation by construction

Motivation

- No preterms and typability relations $\Rightarrow$ directly well-typed objects defined inductively
- No type preservation theorems needed $\Rightarrow$ preservation by construction
- Typed metaprogramming

Motivation

- No preterms and typability relations $\Rightarrow$ directly well-typed objects defined inductively
- No type preservation theorems needed $\Rightarrow$ preservation by construction
- Typed metaprogramming

- We start with a representation of simply typed λ-calculus

Motivation

- No preterms and typability relations $\Rightarrow$ directly well-typed objects defined inductively
- No type preservation theorems needed $\Rightarrow$ preservation by construction
- Typed metaprogramming

- We start with a representation of simply typed λ-calculus
- Then we introduce Quotient Inductive Types (QITs)

Motivation

- No preterms and typability relations $\Rightarrow$ directly well-typed objects defined inductively
- No type preservation theorems needed $\Rightarrow$ preservation by construction
- Typed metaprogramming

- We start with a representation of simply typed λ-calculus
- Then we introduce Quotient Inductive Types (QITs)
- In the second part, QITs are used to represent dependent types in Type Theory

Simply typed λ-calculus

$$
\begin{array}{lcl}
\textbf{data}\ \mathsf{Ty} & : & \mathsf{Set}\ \textbf{where} \\
\quad \iota & : & \mathsf{Ty} \\
\quad \_ \Rightarrow \_ & : & \mathsf{Ty} \to \mathsf{Ty} \to \mathsf{Ty} \\
\textbf{data}\ \mathsf{Con} & : & \mathsf{Set}\ \textbf{where} \\
\quad \bullet & : & \mathsf{Con} \\
\quad \_,\_ & : & \mathsf{Con} \to \mathsf{Ty} \to \mathsf{Con} \\
\textbf{data}\ \mathsf{Var} & : & \mathsf{Con} \to \mathsf{Ty} \to \mathsf{Set}\ \textbf{where} \\
\quad \mathsf{zero} & : & \mathsf{Var}\ (\Gamma\ ,\ \sigma)\ \sigma \\
\quad \mathsf{suc} & : & \mathsf{Var}\ \Gamma\ \sigma \to \mathsf{Var}\ (\Gamma\ ,\ \tau)\ \sigma \\
\textbf{data}\ \mathsf{Tm} & : & \mathsf{Con} \to \mathsf{Ty} \to \mathsf{Set}\ \textbf{where} \\
\quad \mathsf{var} & : & \mathsf{Var}\ \Gamma\ \sigma \to \mathsf{Tm}\ \Gamma\ \sigma \\
\quad \_@\_ & : & \mathsf{Tm}\ \Gamma\ (\sigma \Rightarrow \tau) \to \mathsf{Tm}\ \Gamma\ \sigma \to \mathsf{Tm}\ \Gamma\ \tau \\
\quad \Lambda & : & \mathsf{Tm}\ (\Gamma\ ,\ \sigma)\ \tau \to \mathsf{Tm}\ \Gamma\ (\sigma \Rightarrow \tau)
\end{array}
$$

de Bruijn indices

$$\lambda x.x \quad \longrightarrow \quad \lambda\, 0$$

de Bruijn indices

$$\begin{aligned} \lambda x.x &\quad\longrightarrow\quad \lambda\, 0 \\ \lambda x.\lambda y.\lambda z.x\, z\, (y\, z) &\quad\longrightarrow\quad \lambda\, \lambda\, \lambda\, 2\, 0\, (1\, 0) \end{aligned}$$

de Bruijn indices

$$\begin{aligned}
\lambda x.x &\longrightarrow \lambda\, 0 \\
\lambda x.\lambda y.\lambda z.x\, z\, (y\, z) &\longrightarrow \lambda\, \lambda\, \lambda\, 2\, 0\, (1\, 0) \\
\lambda z.(\lambda y.y\, (\lambda x.x))(\lambda x.z\, x) &\longrightarrow \lambda\, (\lambda\, 0\, (\lambda\, 0))(\lambda\, 1\, 0)
\end{aligned}$$

Simply typed λ-calculus

$$\mathsf{I} : \mathsf{Tm}\ \bullet\ (\iota \Rightarrow \iota)$$
$$\mathsf{I} = \Lambda\ (\mathit{var}\ \ \mathit{zero})$$

Simply typed λ-calculus

$$\mathsf{I} : \mathsf{Tm}\ \bullet\ (\iota \Rightarrow \iota)$$
$$\mathsf{I} = \Lambda\ (\mathit{var}\ \mathit{zero})$$

$$\mathsf{K} : \mathsf{Tm}\ \bullet\ (\iota \Rightarrow (\iota \Rightarrow \iota))$$
$$\mathsf{K} = \Lambda\ (\Lambda\ (\mathit{var}\ \mathit{zero}))$$

Simply typed λ-calculus

$$\mathsf{I} : \mathsf{Tm}\ \bullet\ (\iota \Rightarrow \iota)$$
$$\mathsf{I} = \Lambda\ (\mathit{var}\ \mathit{zero})$$

$$\mathsf{K} : \mathsf{Tm}\ \bullet\ (\iota \Rightarrow (\iota \Rightarrow \iota))$$
$$\mathsf{K} = \Lambda\ (\Lambda\ (\mathit{var}\ \mathit{zero}))$$

$$\mathsf{I}_{\mathsf{again}} : \mathsf{Tm}\ \bullet\ (\iota \Rightarrow \iota)$$
$$\mathsf{I}_{\mathsf{again}} = \Lambda\ (\Lambda\ (\mathit{var}\ (\mathit{suc}\ \mathit{zero}))\ @\ \mathit{var}\ \mathit{zero})$$

Inductive-inductive types

```
data Con : Set
data Ty  : Con → Set

data Con where
  •   : Con
  _,_ : (Γ : Con) → Ty Γ → Con

data Ty where
  U   : ∀ {Γ} → Ty Γ
  Π   : ∀ {Γ} (A : Ty Γ) (B : Ty (Γ , A)) → Ty Γ
```

Inductive-inductive types

```
data Con : Set
data Ty  : Con → Set

data Con where
  •     : Con
  _,_   : (Γ : Con) → Ty Γ → Con

data Ty where
  U     : ∀ {Γ} → Ty Γ
  Π     : ∀ {Γ} (A : Ty Γ) (B : Ty (Γ , A)) → Ty Γ
```

```
module RecConTy where
  record Motives : Set₁ where
    field
      Conᴹ  : Set
      Tyᴹ   : Conᴹ → Set
  record Methods (M : Motives) : Set₁ where
    open Motives M
    field
      •ᴹ     : Conᴹ
      _,Cᴹ_  : (Γᴹ : Conᴹ) → Tyᴹ Γᴹ → Conᴹ
      Uᴹ     : {Γᴹ : Conᴹ} → Tyᴹ Γᴹ
      Πᴹ     : {Γᴹ : Conᴹ} (Aᴹ : Tyᴹ Γᴹ)
               (Bᴹ : Tyᴹ (Γᴹ ,Cᴹ Aᴹ)) → Tyᴹ Γᴹ
  module rec (M : Motives) (m : Methods M) where
    open Motives M
    open Methods m

    RecCon : Con → Conᴹ
    RecTy : {Γ : Con} (A : Ty Γ) → Tyᴹ (RecCon Γ)

    RecCon •       = •ᴹ
    RecCon (Γ , A) = RecCon Γ ,Cᴹ RecTy A

    RecTy  U       = Uᴹ
    RecTy  (Π A B) = Πᴹ (RecTy A) (RecTy B)
```

Inductive-inductive types

```
data Con : Set
data Ty  : Con → Set

data Con where
  •     : Con
  _,_   : (Γ : Con) → Ty Γ → Con

data Ty where
  U     : ∀ {Γ} → Ty Γ
  Π     : ∀ {Γ} (A : Ty Γ) (B : Ty (Γ , A)) → Ty Γ
```

```
module RecConTy where
  record Motives : Set₁ where
    field
      Conᴹ  : Set
      Tyᴹ   : Conᴹ → Set
  record Methods (M : Motives) : Set₁ where
    open Motives M
    field
      •ᴹ     : Conᴹ
      _,Cᴹ_  : (Γᴹ : Conᴹ) → Tyᴹ Γᴹ → Conᴹ
      Uᴹ     : {Γᴹ : Conᴹ} → Tyᴹ Γᴹ
      Πᴹ     : {Γᴹ : Conᴹ} (Aᴹ : Tyᴹ Γᴹ)
               (Bᴹ : Tyᴹ (Γᴹ ,Cᴹ Aᴹ)) → Tyᴹ Γᴹ
  module rec (M : Motives) (m : Methods M) where
    open Motives M
    open Methods m

    RecCon : Con → Conᴹ
    RecTy : {Γ : Con} (A : Ty Γ) → Tyᴹ (RecCon Γ)

    RecCon •       = •ᴹ
    RecCon (Γ , A) = RecCon Γ ,Cᴹ RecTy A

    RecTy U        = Uᴹ
    RecTy (Π A B)  = Πᴹ (RecTy A) (RecTy B)
```

Motive explaining what is to be achieved by elimination
Methods explaining how the motive is to be pursued for each constructor in turn

Quotient types

$$
\begin{array}{l}
\textbf{data}\ \mathsf{T}_0\ :\ \mathsf{Set}\ \textbf{where} \\
\quad \mathsf{leaf}\ :\ \mathsf{T}_0 \\
\quad \mathsf{node}\ :\ (\mathbb{N} \to \mathsf{T}_0) \to \mathsf{T}_0
\end{array}
$$

Quotient types

$$
\begin{array}{l}
\textbf{data}\ \mathsf{T}_0 : \mathsf{Set}\ \textbf{where} \\
\quad \mathsf{leaf} : \mathsf{T}_0 \\
\quad \mathsf{node} : (\mathbb{N} \to \mathsf{T}_0) \to \mathsf{T}_0
\end{array}
$$

$$
\begin{array}{lll}
\textbf{data}\ \_\tilde{}\_ & : & \mathsf{T}_0 \to \mathsf{T}_0 \to \mathsf{Set}\ \textbf{where} \\
\quad \mathsf{leaf} & : & \mathsf{leaf} \sim \mathsf{leaf} \\
\quad \mathsf{node} & : & \{\mathsf{f}\ \mathsf{g} : \mathbb{N} \to \mathsf{T}_0\} \to (\forall\ \{\mathsf{n}\} \to \mathsf{f}\ \mathsf{n} \sim \mathsf{g}\ \mathsf{n}) \\
& \to & \mathsf{node}\ \mathsf{f} \sim \mathsf{node}\ \mathsf{g} \\
\quad \mathsf{perm} & : & (\mathsf{g} : \mathbb{N} \to \mathsf{T}_0)\ (\mathsf{f} : \mathbb{N} \to \mathbb{N}) \to \mathsf{isIso}\ \mathsf{f} \\
& \to & \mathsf{node}\ \mathsf{g} \sim \mathsf{node}\ (\mathsf{g} \circ \mathsf{f})
\end{array}
$$

Quotient types

$$
\begin{aligned}
&\textbf{data}\ \mathsf{T}_0\ :\ \mathsf{Set}\ \textbf{where}\\
&\quad \mathsf{leaf}\ :\ \mathsf{T}_0\\
&\quad \mathsf{node}\ :\ (\mathbb{N} \to \mathsf{T}_0) \to \mathsf{T}_0
\end{aligned}
$$

$$
\begin{array}{lcl}
\textbf{data}\ \_\tilde{}\_ & : & \mathsf{T}_0 \to \mathsf{T}_0 \to \mathsf{Set}\ \textbf{where}\\
\quad \mathsf{leaf} & : & \mathsf{leaf} \sim \mathsf{leaf}\\
\quad \mathsf{node} & : & \{\mathsf{f}\ \mathsf{g}\ :\ \mathbb{N} \to \mathsf{T}_0\} \to (\forall\ \{\mathsf{n}\} \to \mathsf{f}\ \mathsf{n} \sim \mathsf{g}\ \mathsf{n})\\
 & \to & \mathsf{node}\ \mathsf{f} \sim \mathsf{node}\ \mathsf{g}\\
\quad \mathsf{perm} & : & (\mathsf{g}\ :\ \mathbb{N} \to \mathsf{T}_0)\ (\mathsf{f}\ :\ \mathbb{N} \to \mathbb{N}) \to \mathsf{isIso}\ \mathsf{f}\\
 & \to & \mathsf{node}\ \mathsf{g} \sim \mathsf{node}\ (\mathsf{g} \circ \mathsf{f})
\end{array}
$$

$$
\begin{aligned}
\mathsf{T}\ &:\ \mathsf{Set}\\
\mathsf{T}\ &=\ \mathsf{T}_0\ /\ \_\tilde{}\_
\end{aligned}
$$

Quotient types

```
data _/_ (A : Set) (R : A → A → Set) : Set where
  [_] : A → A / R
postulate
  [_]≡ :  ∀ {A} {R : A → A → Set} {a b : A}
       → R a b → [ a ] ≡ [ b ]
module Elim_/_
  (A : Set) (R : A → A → Set)
  (Qᴹ : A / R → Set)
  ([_]ᴹ : (a : A) → Qᴹ [ a ])
  ([_]≡ᴹ :   {a b : A} (r : R a b)
          → [ a ]ᴹ ≡[ ap Qᴹ [ r ]≡ ]≡ [ b ]ᴹ)
  where
    Elim : (x : A / R) → Qᴹ x
    Elim [ x ] = [ x ]ᴹ
```

Quotient inductive types

Quotient inductive types are HITs with only strict equality (no higher paths)

Quotient inductive types

Quotient inductive types are HITs with only strict equality (no higher paths)

$$
\begin{array}{lll}
\textbf{data}\ \mathsf{T} & : & \mathsf{Set}\ \textbf{where} \\
\quad \mathsf{leaf} & : & \mathsf{T} \\
\quad \mathsf{node} & : & (\mathbb{N} \to \mathsf{T}) \to \mathsf{T} \\
\textbf{postulate} & & \\
\quad \mathsf{perm} & : & (\mathsf{g} : \mathbb{N} \to \mathsf{T})\ (\mathsf{f} : \mathbb{N} \to \mathbb{N}) \to \mathsf{isIso}\ \mathsf{f} \\
 & \to & \mathsf{node}\ \mathsf{g} \equiv \mathsf{node}\ (\mathsf{g} \circ \mathsf{f})
\end{array}
$$

$$
\begin{array}{lll}
\textbf{module}\ \mathsf{ElimT} & & \\
\quad (\mathsf{T}^{\mathsf{M}} & : & \mathsf{T} \to \mathsf{Set}) \\
\quad (\mathsf{leaf}^{\mathsf{M}} & : & \mathsf{T}^{\mathsf{M}}\ \mathsf{leaf}) \\
\quad (\mathsf{node}^{\mathsf{M}} & : & \{\mathsf{f} : \mathbb{N} \to \mathsf{T}\}\ (\mathsf{f}^{\mathsf{M}} : (\mathsf{n} : \mathbb{N}) \to \mathsf{T}^{\mathsf{M}}\ (\mathsf{f}\ \mathsf{n})) \\
 & & \to \mathsf{T}^{\mathsf{M}}\ (\mathsf{node}\ \mathsf{f})) \\
\quad (\mathsf{perm}^{\mathsf{M}} & : & \{\mathsf{g} : \mathbb{N} \to \mathsf{T}\}\ (\mathsf{g}^{\mathsf{M}} : (\mathsf{n} : \mathbb{N}) \to \mathsf{T}^{\mathsf{M}}\ (\mathsf{g}\ \mathsf{n})) \\
 & & (\mathsf{f} : \mathbb{N} \to \mathbb{N})\ (\mathsf{p} : \mathsf{isIso}\ \mathsf{f}) \\
 & & \to \mathsf{node}^{\mathsf{M}}\ \mathsf{g}^{\mathsf{M}} \equiv[\ \mathsf{ap}\ \mathsf{T}^{\mathsf{M}}\ (\mathsf{perm}\ \mathsf{g}\ \mathsf{f}\ \mathsf{p})\ ]\equiv \\
 & & \mathsf{node}^{\mathsf{M}}\ (\mathsf{g}^{\mathsf{M}} \circ \mathsf{f})) \\
\quad \textbf{where} & &
\end{array}
$$

$$
\begin{array}{l}
\quad \mathsf{Elim} : (\mathsf{t} : \mathsf{T}) \to \mathsf{T}^{\mathsf{M}}\ \mathsf{t} \\
\quad \mathsf{Elim}\ \mathsf{leaf} = \mathsf{leaf}^{\mathsf{M}} \\
\quad \mathsf{Elim}\ (\mathsf{node}\ \mathsf{f}) = \mathsf{node}^{\mathsf{M}}\ (\lambda\ \mathsf{n} \to \mathsf{Elim}\ (\mathsf{f}\ \mathsf{n}))
\end{array}
$$