Universe Polymorphism Expressed as a Rewriting System

Guillaume Genestier

Thursday February 27th, 2020

Contents

Good Old PTS

A *Pure Type System* (PTS) is defined by:

- A set $\mathcal{S}$, representing sorts (families of objects),
- $\mathcal{A} \subseteq \mathcal{S}^2$, declaring membership between sorts,
- $\mathcal{R} \subseteq \mathcal{S}^3$, stating over which sorts one can quantify, and in which sort the result inhabits.

PTS's of interest

Edimburgh Logical Framework (LF)

The simplest PTS featuring *Dependent types*.

$$\mathcal{S} = \{\star, \square\} \qquad \mathcal{A} = \{\star : \square\} \qquad \mathcal{R} = \{(\star, \star, \star); (\star, \square, \square)\}$$

Terms in LF

$$\begin{aligned} 1 &: \mathbb{N} : \star \\ fact &: \mathbb{N} \to \mathbb{N} : \star \\ Vec &: \mathbb{N} \to \star : \square \\ cons &: El \to (n : \mathbb{N}) \to Vec\, n \to Vec\,(n+1) \end{aligned}$$

Infinite Predicative Hierarchy

$$\mathcal{S} = \{\ \mathrm{Set}_i \mid i \in \mathbb{N}\} \qquad \mathcal{A} = \{\mathrm{Set}_i : \mathrm{Set}_{i+1}\}_i$$
$$\mathcal{R} = \left\{(\mathrm{Set}_i, \mathrm{Set}_j, \mathrm{Set}_{\max(i,j)})\right\}_{i,j}$$

$\texttt{Univ}_\square : \star.$ $\qquad$ $\texttt{Univ}_\star : \star.$

$\texttt{Term}_\square : \texttt{Univ}_\square \Rightarrow \star.$ $\qquad$ $\texttt{Term}_\star : \texttt{Univ}_\star \Rightarrow \star.$

$\texttt{code}_{\star\square} : \texttt{Univ}_\square.$ $\qquad$ $\texttt{Term}_\square\ \texttt{code}_{\star\square} \longrightarrow \texttt{Univ}_\star.$

$\texttt{prod}_{\star\star\star} : (\texttt{A} : \texttt{Univ}_\star) \Rightarrow (\texttt{Term}_\star\ \texttt{A} \Rightarrow \texttt{Univ}_\star) \Rightarrow \texttt{Univ}_\star.$

$\texttt{Term}_\star\ (\texttt{prod}_{\star\star\star}\ \texttt{A}\ \texttt{B}) \longrightarrow (\texttt{x} : \texttt{Term}_\star\ \texttt{A}) \Rightarrow \texttt{Term}_\star\ (\texttt{B}\ \texttt{x}).$

$\texttt{prod}_{\star\square\square} : (\texttt{A} : \texttt{Univ}_\star) \Rightarrow (\texttt{Term}_\star\ \texttt{A} \Rightarrow \texttt{Univ}_\square) \Rightarrow \texttt{Univ}_\square.$

$\texttt{Term}_\square\ (\texttt{prod}_{\star\square\square}\ \texttt{A}\ \texttt{B}) \longrightarrow (\texttt{x} : \texttt{Term}_\star\ \texttt{A}) \Rightarrow \texttt{Term}_\square\ (\texttt{B}\ \texttt{x}).$

A Great Encoding

Theorem (Soundness of the encoding)

If P is a functional PTS and $\Gamma \vdash t : A$ in P, then $\|\Gamma\| \vdash_{LF} |t| : \|A\|$ in LF.

Theorem (Conservativity of the encoding)

Let P be a functional PTS. For all Γ context and A term of P, if there is a normal term t of LF such that $\|\Gamma\| \vdash_{LF} t : \|A\|$, then there is a term u of P such that $\Gamma \vdash u : A$ in P.

What if $\mathcal{S}$ is infinite [Assaf, 2015]

```
𝒮 : ⋆.                     Set : Lvl ⇒ 𝒮.

def ax : 𝒮 ⇒ 𝒮.
ax (Set  i) ⟶ ax (succ i).

def rule : 𝒮 ⇒ 𝒮 ⇒ 𝒮.
rule (Set  i) (Set  j) ⟶ Set (max i j).

Univ : (s : 𝒮) ⇒ ⋆.
Term : (s : 𝒮) ⇒ Univ s ⇒ ⋆.

prod : (s1 : 𝒮) ⇒ (s2 : 𝒮) ⇒ (a : Univ s1) ⇒
       (Term s1 a ⇒ Univ s2) ⇒ Univ (rule s1 s2).

Term _ (prod s1 s2 a b) ⟶
         (x : Term s1 a) ⇒ Term s2 (b x).
```

Contents

We want more

Example

How to encode the polymorphic type of lists?

We want more

Example

How to encode the polymorphic type of lists?

```
inductive datatype List (A : Set₀) : Set₀
  []    : List A
  _::_  : A ⇒ List A ⇒ List A
```

We want more

Example

How to encode the polymorphic type of lists?

```
inductive datatype List (A : Set₀) : Set₀
  []    : List A
  _::_  : A ⇒ List A ⇒ List A
```

The day we want a list of functions from $\mathbb{N}$ to T_0:

```
inductive datatype List' (A : Set₁) : Set₁
  []'    : List' A
  _::'_  : A ⇒ List' A ⇒ List' A
```

We want more

Example

How to encode the polymorphic type of lists?

```
inductive datatype List (A : Set₀) : Set₀
  []    : List A
  _::_ : A ⇒ List A ⇒ List A
```

The day we want a list of functions from $\mathbb{N}$ to T_0:

```
inductive datatype List' (A : Set₁) : Set₁
  []'    : List' A
  _::'_ : A ⇒ List' A ⇒ List' A
```

Reaction

Perfect: Computers are good at copy-pasting and global-replacing!

Zero Problem, One Solution

Reaction

What about universal quantification?

```
inductive datatype List (ℓ : Level) (A : Setℓ) : Setℓ
  []    : List A
  _::_ : A ⇒ List A ⇒ List A
```

Not in the syntax of *Pure Type System*.

Uniform Universe Polymorphic Type System

Sorts

$\mathcal{S} = \mathbb{H} \times \mathbb{L}$

Axioms

For all $s \in \mathbb{H}$, if there are $s' \in \mathbb{H}$ and $\ell, \ell' \in \mathbb{L}$ such that $(s_\ell : s'_{\ell'}) \in \mathcal{A}$, then there is a total function $a_{s,s'} : \mathbb{L} \to \mathbb{L}$ such that $\left\{ (s_\ell : s'_{a_{s,s'}(\ell)}) \middle| \ell \in \mathbb{L} \right\} \subseteq \mathcal{A}$.

Same for Rules.

Enriched Type System

Context

The global signature $f : \forall[\ell_1, \ldots, \ell_n].A$

The level variables ℓ

The local context $x : A$

$$\Sigma; \Theta; \Gamma$$

(var) $$\frac{\Sigma; \Theta; \Gamma \vdash A : s_\gamma}{\Sigma; \Theta; \Gamma, x : A \vdash x : A}\; x \notin \Sigma, \Gamma$$

(sig) $$\frac{\Sigma; \Theta; [] \vdash A : s_\gamma}{\Sigma, x : \forall\Theta.A; \Theta'; [] \vdash x : \forall\Theta.A}\; x \notin \Sigma, \Gamma$$

(inst) $$\frac{\Sigma; \Theta; \Gamma \vdash t : \forall[i_1, \ldots, i_n], A \quad \Theta \vdash \gamma_1 \ \text{isLvl} \quad \ldots \quad \Theta \vdash \gamma_n \ \text{isLvl}}{\Sigma; \Theta; \Gamma \vdash t[\gamma_1, \ldots, \gamma_n] : A \left[\gamma_k / i_k\right]_k}$$

Contents

One Solution, New Problem

Question

What is the type of $\forall \ell, \mathsf{Set}_\ell$?

One Solution, New Problem

Question

What is the type of $\forall \ell, \mathsf{Set}_\ell$?

A brand new sort: Sort_ω

- Not typable,
- Type of no sorts,
- On which we cannot quantify,
- For internal use only, not in the syntax.

Encoding it

```
Term _ (prod s1 s2 a b) ⟶
           (x : Term s1 a) ⇒ Term s2 (b x).
```

$\texttt{Sort}_\omega$: $\mathcal{S}$.

$\forall_{Lvl}$: s: (Lvl ⇒ $\mathcal{S}$) ⇒
      (l : Lvl ⇒ Univ (f l)) ⇒
      Univ $\texttt{Sort}_\omega$.

Term _ ($\forall_{Lvl}$ s b) ⟶ (l : Lvl) ⇒ Term (s l) (b l).

Example ($\forall \ell, \mathrm{Set}_\ell$)

$\forall_{Lvl}$ (λl, ax (Set l)) (λl, code (Set l)).

Translation

Definition (Translation of sorts)

$\langle \mathrm{Sort}_\omega \rangle = \mathtt{Sort}_\omega; \qquad \langle \mathrm{Set}_\ell \rangle = \mathtt{Set}\ |\ell|_L.$

Definition (Translation of terms)

A well-typed term is translated by:

$|x| = \mathtt{x}; \qquad \left|\lambda x^A . t\right| = \lambda x : \|A\| \,.\, |t|; \qquad |\mathrm{Set}_\ell| = \mathtt{code}\ \langle \mathrm{Set}_\ell \rangle;$

$|(x : A) \to B| = \mathtt{prod}\ \langle s_A \rangle\ \langle s_B \rangle\ |A|\ (\lambda x : \|A\| \,.\, |B|);$

$|\forall \ell, A| = \forall_{Lvl}\ (\lambda \ell : \mathtt{Level}. \langle s_A \rangle)\ (\lambda \ell : \mathtt{Level}.\, |A|).$

Definition (Translation of types)

A well-typed term T is translated as type by:

$\|T\| = \mathtt{Term}\ \langle s_T \rangle\ \ |T|$

A Pretty Good Encoding

Theorem (Soundness of the encoding)

Assuming $|.|_L$ is such that equality of levels implies convertibility of their translations.
If P is a functional uniform universe polymorphic PTS, then $\Gamma \vdash t : A$ in P implies $\|\Gamma\| \vdash_{LF} |t| : \|A\|$ in LF.

Contents

Predicative Naturals

Grammar of universe levels

$$t, u ::= x \in \mathcal{X} \mid 0 \mid \mathrm{s}\, t \mid \max\, t\, u$$

Our goal

$$t \leftrightsquigarrow^* u \text{ if and only if } \forall \sigma : \mathcal{X} \to \mathbb{N}, [\![t]\!]_\sigma = [\![u]\!]_\sigma$$

The problems

For all $n > m$ and all σ,

$$[\![\max\,(s^n\, x)\,(s^m\, x)]\!]_\sigma = [\![s^n\, x]\!]_\sigma \qquad [\![\max\,(s^n\, x)\,(s^m\, 0)]\!]_\sigma = [\![s^n\, x]\!]_\sigma$$

We do not want an infinity of (non-linear) rewrite rules.

A New Hope

Reasoning Modulo AC

- for all t, t has a unique normal form (modulo associativity and commutativity),
- for all t and u in $\mathcal{L}$,

$$t\!\downarrow\ \equiv_{AC}\ u\!\downarrow \text{ if and only if } \quad \forall\sigma : \mathcal{X} \to \mathbb{N}, [\![t]\!]_\sigma = [\![u]\!]_\sigma$$

Normal Forms

$$\mathrm{Max}\ i\ \ \{j_1 + x_1, j_2 + x_2, ...\}$$

where:

- $i, j_1, j_2, \ldots$ are closed natural numbers,
- $x_1, x_2, \ldots$ are distinct variables,
- for all k, $i \geqslant j_k$.

Two Distinct Types

The Original Syntax

```
Level : ⋆.

0 : Level.
s : Level ⇒ Level.
max: Level ⇒ Level ⇒ Level.
```

Unary Naturals

$\mathbb{N}$: ⋆. $0_{\mathbb{N}}$: $\mathbb{N}$. $s_{\mathbb{N}}$: $\mathbb{N} \Rightarrow \mathbb{N}$.

definition $1_{\mathbb{N}}$:= $s_{\mathbb{N}}$ $0_{\mathbb{N}}$.

$max_{\mathbb{N}}$: $\mathbb{N} \Rightarrow \mathbb{N} \Rightarrow \mathbb{N}$. $\_+_{\mathbb{N}}\_$: $\mathbb{N} \Rightarrow \mathbb{N} \Rightarrow \mathbb{N}$.

Sets Using Rewriting

Signature

```
LvlSet : ⋆.

∅ : LvlSet.
{_⊕_} : ℕ ⇒ Level ⇒ LvlSet.
associative-commutative _∪_ on LvlSet.
```

Rules on Union

```
x ∪ ∅ ⟶ x.
{i ⊕ l} ∪ {j ⊕ l} ⟶ {(max_ℕ i j) ⊕ l}.
```

The Smart Constructor Max

Signature

```
Max : ℕ ⇒ LvlSet ⇒ Level.

mapPlus : ℕ ⇒ LvlSet ⇒ LvlSet.
```

Distribution Rules

```
Max i (j (Max k l))                  ⟶
        Max (maxℕ i (j +ℕ k)) (mapPlus j l).
Max i ({j ⊕ (Max k l)} ∪ tl) ⟶
     Max (maxℕ i (j +ℕ k)) ((mapPlus j l) ∪ tl).
```

The Final Part

Implementation of the Syntax

$$\begin{array}{lll} \texttt{0} & \longrightarrow & \texttt{Max}\ \texttt{0}_{\mathbb{N}}\ \emptyset. \\ \texttt{(s x)} & \longrightarrow & \texttt{Max}\ \texttt{1}_{\mathbb{N}}\ \{\texttt{1}_{\mathbb{N}} \oplus \texttt{x}\}. \\ \texttt{(max x y)} & \longrightarrow & \texttt{Max}\ \texttt{0}_{\mathbb{N}}\ (\{\texttt{0}_{\mathbb{N}} \oplus \texttt{x}\} \cup \{\texttt{0}_{\mathbb{N}} \oplus \texttt{y}\}). \end{array}$$

Proposition

The absence of variable of type $\mathbb{N}$ or `LvlSet` *ensures the uniqueness of normal form (modulo AC) property.*

Universe Polymorphism Expressed as a Rewriting System

Guillaume Genestier

Thursday February 27th, 2020