Resizing Prop down to an axiom

Stefan Monnier

monnier@iro.umontreal.ca

Université de Montréal

Impredicativity 101

According to the O.E.D:

> im- + predicative, adj. & n.: With a sneaky form of circularity
>
> $\sim$1389, Chaucer: *Can't trust this dude, he's too impredicative*!

The origin of the notion of *types*, from Russell:

> Let S be the set of all sets that do not contain themselves:
>
> Does S contain itself?

Fix: introduce a stratification to prevent such self-applications

Anti-fix: some forms of impredicativity seem consistent and useful

Why do I care?

Working on Typer, an ML/Haskell with dependent types and macros

Typer: low-level λ-calculus intermediate language

Impredicativity used in:

- Encoding of modules into tuples
  (containing level-polymorphic definitions)
- Closure conversion
- The desire to subsume System-F

Existing forms of impredicativity don't seem sufficient

Prop is an irregularity in universe polymorphism

Forms of impredicativity

Impredicative universes, most common of them is Prop:

$$\tau_2 : \mathsf{Prop} \quad\Longrightarrow\quad (x : \tau_1) \to \tau_2 : \mathsf{Prop}$$

As present in System-F, Coq, and many others

Resizing axioms:

$$\tau : \mathsf{Type}_\ell \wedge P(\tau) \quad\Longrightarrow\quad \tau : \mathsf{Type}_{\ell'}$$

Most famously, HoTT's propositional resizing

Impredicative universe polymorphism:

$$\tau_2 : \mathsf{Type}_\ell \quad\Longrightarrow\quad (l : \mathsf{Level}) \equiv\!> \tau_2 : \mathsf{Type}_{\ell[0/l]}$$

Foolish experiment in Typer, proposed in last episode

The Question

How do forms of impredicativity relate?

$\Downarrow$

How do impredicative universes compare to resizing axioms?

$\Downarrow$

How does impredicative Prop compare to HoTT propositional resizing?

$\Downarrow$

Show equivalence of close(?) relatives to those systems

Université de Montréal

Base calculus: pCCω

We start from a predicative PTS with a tower of universes:

$$e, \tau \quad ::= \quad s \mid x \mid (x : \tau_1) \to \tau_2 \mid \lambda x : \tau . e \mid e_1\ e_2$$

$$\begin{array}{rcll}
\mathcal{S} & = & \{\ \mathsf{Type}_\ell & \mid \ell \in \mathbb{N}\ \} \\
\mathcal{A} & = & \{\ (\mathsf{Type}_\ell : \mathsf{Type}_{\ell+1}) & \mid \ell \in \mathbb{N}\ \} \\
\mathcal{R} & = & \{\ (\mathsf{Type}_{\ell_1}, \mathsf{Type}_{\ell_2}, \mathsf{Type}_{\ell_1 \sqcup \ell_2}) & \mid \ell_1, \ell_2 \in \mathbb{N}\ \}
\end{array}$$

Impredicative universe: iCCω

We add impredicative quantification to pCCω's bottom universe:

$$e, \tau \;\; ::= \;\; s \mid x \mid (x : \tau_1) \to \tau_2 \mid \lambda x : \tau . e \mid e_1 \; e_2$$

$$\begin{array}{lcll}
\mathcal{S} & = & \{ \, \mathsf{Type}_\ell & \mid \ell \in \mathbb{N} \, \} \\
\mathcal{A} & = & \{ \, (\mathsf{Type}_\ell : \mathsf{Type}_{\ell+1}) & \mid \ell \in \mathbb{N} \, \} \\
\mathcal{R} & = & \{ \, (\mathsf{Type}_{\ell_1}, \mathsf{Type}_{\ell_2}, \mathsf{Type}_{\ell_1 \sqcup \ell_2}) & \mid \ell_1, \ell_2 \in \mathbb{N} \, \} \\
 & & \cup \, \{ \, (\mathsf{Type}_\ell, \mathsf{Type}_0, \mathsf{Type}_0) & \mid \ell \in \mathbb{N} \wedge \ell > 0 \, \}
\end{array}$$

Resizing axiom: rCCω

We start from an erasure monad

A combination of HoTT's propositional truncation with its resizing axiom:

$$
\begin{array}{lcll}
||\cdot|| & : & \mathsf{Type}_\ell \to \mathsf{Type}_0 & (\forall \ell) \\
|\cdot| & : & (t : \mathsf{Type}_\ell) \to t \to ||t|| & (\forall \ell) \\
\mathsf{bind} & : & (t_1 : \mathsf{Type}_{\ell_1}) \to (t_2 : \mathsf{Type}_{\ell_2}) & (\forall \ell_1, \ell_2) \\
 & & \to ||t_1|| \to (t_1 \to ||t_2||) \to ||t_2|| &
\end{array}
$$

$$
\Gamma \vdash \mathsf{bind}_{\tau_1,\tau_2}\ |e_1|\ \lambda x : \tau_1 . e_2 \ \simeq\ e_2[e_1/x]\ : \tau_2
$$

Similar monad used by Arnaud Spiwack to axiomatize Hurkens's paradox

Warm up: translating rCCω to iCCω

Encoding rCCω into iCCω is easy. We can just provide definitions:

$$\begin{array}{lcl} ||\tau|| & = & (t : \mathsf{Type}_0) \to (\tau \to t) \to t \\ |e|_\tau & = & \lambda t : \mathsf{Type}_0 . \lambda k : (\tau \to t) . k\ e \\ \mathsf{bind}_{\tau_1,\tau_2}\ e_1\ e_2 & = & e_1\ ||\tau_2||\ e_2 \end{array}$$

And indeed it satisfies the desired reduction:

$$\begin{array}{l} \mathsf{bind}\ |e_1|\ \lambda x : \tau_1 . e_2 \\ \simeq\ \ (\lambda \alpha : \mathsf{Type}_0 . \lambda f : (\tau_1 \to \alpha) . f\ e_1)\ \tau_2\ (\lambda x : \tau_1 . e_2) \\ \simeq\ \ e_2[e_1/x] \end{array}$$

Encoding Prop using erasure

We want to encode (written $[\![\cdot]\!]$) iCCω terms into rCCω

Core problem: $(x:\tau_1) \to \tau_2$ where τ_1 : $\mathsf{Type}_{>0}$ and τ_2 : Type_0

- iCCω puts them in Type_0
- rCCω would put them in Type_ℓ

Basic approach: use $||\cdot||$ to bring them down to Type_0:

$$[\![(x:\tau_1) \to \tau_2]\!] = ||(x:[\![\tau_1]\!]) \to [\![\tau_2]\!]||$$

Inspired by Coq, we erase all Type_0 terms

Université de Montréal

Translating iCCω to rCCω: first try

$$
\begin{array}{lcll}
[x] & = & x & \\
[\mathsf{Type}_\ell] & = & \mathsf{Type}_\ell & \\
[(x:\tau_1) \to \tau_2] & = & \begin{cases} ||(x:[\tau_1]) \to [\tau_2]|| & \text{if in the Type}_0\text{ universe} \\ (x:[\tau_1]) \to [\tau_2] & \text{otherwise} \end{cases} \\
[\lambda x:\tau.e] & = & \begin{cases} |\lambda x:[\tau].[e]| & \text{if in the Type}_0\text{ universe} \\ \lambda x:[\tau].[e] & \text{otherwise} \end{cases} \\
[e_1\ e_2] & = & \begin{cases} \mathsf{bind}\ [e_1]\ \lambda f.f\ [e_2] & \text{if } e_1 \text{ in the Type}_0\text{ universe} \\ [e_1]\ [e_2] & \text{otherwise} \end{cases}
\end{array}
$$

¿Type non-preservation?

When we try to show $\Gamma \vdash e : \tau \implies [\Gamma] \vdash [e] : [\tau]$

¡We fail on bind $[e_1]\ \lambda f.f\ [e_2]$!

- $f\ [e_2]$ has type $[\tau_2]$
- We know that τ_2 : Type$_0$ so *we* know $[\tau_2]$ *will be* erased
- But the type does not reflect that unless it's a literal arrow

¿Type non-preservation?

When we try to show $\Gamma \vdash e : \tau \implies [\Gamma] \vdash [e] : [\tau]$

¡We fail on bind $[e_1]\ \lambda f. f\ [e_2]$!

- $f\ [e_2]$ has type $[\tau_2]$
- We know that τ_2 : Type$_0$ so *we* know $[\tau_2]$ *will be* erased
- But the type does not reflect that unless it's a literal arrow

$\implies$ Weaken bind so it does not require literally $||\tau||$

Instead, require a proof of erasure

$\implies$ Less of a monad

$\implies$ Also, we need pairs

New rCCω axioms

$$\begin{array}{lcl} \text{bind} & : & (t_1 : \text{Type}_\ell) \to (t_2 : \text{Type}_0) \to \qquad (\forall \ell) \\ & & ||t_1|| \to (t_1 \to \boxed{(t_2 \times \text{IsErased}\ t_2)}) \to \\ & & \boxed{(t_2 \times \text{IsErased}\ t_2)} \end{array}$$

$$\begin{array}{lcl} \text{IsErased} & : & \text{Type}_0 \to \text{Type}_0 \\ \text{iserased} & : & (t : \text{Type}_\ell) \to \text{IsErased}\ ||t|| \qquad (\forall \ell) \\ \cdot \times \cdot & : & \text{Type}_0 \to \text{Type}_0 \to \text{Type}_0 \\ \langle \cdot, \cdot \rangle & : & (t_1 : \text{Type}_0) \to (t_2 : \text{Type}_0) \to t_1 \to t_2 \to (t_1 \times t_2) \\ \cdot.0 & : & (t_1 : \text{Type}_0) \to (t_2 : \text{Type}_0) \to (t_1 \times t_2) \to t_1 \end{array}$$

$$\Gamma \vdash \langle e_1, e_2 \rangle.0 \simeq e_1 : \tau_1$$

Université de Montréal

New translation of iCCω to rCCω

$$\llbracket \tau \rrbracket = \begin{cases} \lceil \tau \rceil \times \textsf{IsErased}\ \lceil \tau \rceil & \text{if } \tau : \text{Type}_0 \\ \lceil \tau \rceil & \text{otherwise} \end{cases}$$

$$\lceil (x : \tau_1) \to \tau_2 \rceil = \begin{cases} ||(x : \llbracket \tau_1 \rrbracket) \to \llbracket \tau_2 \rrbracket|| & \text{if in the Type}_0 \text{ universe} \\ (x : \llbracket \tau_1 \rrbracket) \to \llbracket \tau_2 \rrbracket & \text{otherwise} \end{cases}$$

$$\lceil e_1\ e_2 \rceil = \begin{cases} \textsf{bind}\ \lceil e_1 \rceil.0\ \lambda f.f\ \lceil e_2 \rceil & \text{if } e_1 \text{ in the Type}_0 \text{ universe} \\ \lceil e_1 \rceil\ \lceil e_2 \rceil & \text{otherwise} \end{cases}$$

$$\lceil \lambda x : \tau_1 . e \rceil = \begin{cases} \langle\ |\lambda x : \llbracket \tau_1 \rrbracket . \lceil e \rceil|, \textsf{iserased}\ ((x : \llbracket \tau_1 \rrbracket) \to \llbracket \tau_2 \rrbracket)\ \rangle & \\ \lambda x : \llbracket \tau_1 \rrbracket . \lceil e \rceil & \text{if not in the Type}_0 \text{ universe} \end{cases}$$

Translating new rCCω to iCCω

$$
\begin{aligned}
||\tau|| &= (t : \mathsf{Type}_0) \to (\tau \to t) \to t \\
|e|_\tau &= \lambda t : \mathsf{Type}_0 . \lambda k : (\tau \to t) . k\ e \\
\mathsf{bind}_{\tau_1, \tau_2}\ e_1\ e_2 &= e_1\ (\tau_2 \times \mathsf{IsErased}\ t_2)\ e_2 \\
\\
\mathsf{IsErased}\ \tau &= (t : \mathsf{Type}_0) \to t \to t \\
\mathsf{iserased}\ \tau &= \lambda t : \mathsf{Type}_0 . \lambda x : t . x \\
\tau_1 \times \tau_2 &= \tau_1 \\
\langle e_1, e_2 \rangle &= e_1 \\
e.0 &= e
\end{aligned}
$$

Differences to traditional Prop

iCCω differs from the usual impredicative CCω:

- Both predicative and impredicative functions from Type$_\ell$ to Type$_0$

  Used to simplify the rCCω encoding

  Could be replaced by cummulativity of universes

Comparison to propositional resizing

The axioms of rCCω are satisfied by HoTT's axioms:

$\|\|\tau\|\|$	$\leq$	*propositional truncation* and *resizing*
IsErased	$\leq$	isProp
bind	$\leq$	the recursion principle of *propositional truncation*

They are less general than those of HoTT:

- rCCω's $\|\|\tau\|\|$ does not imply proof irrelevance
- IsErased can be fulfilled only by erasure/truncation rather than by a proof of irrelevance

Université de Montréal

Adding η to rCCω's

$$\Gamma \vdash \lambda x\!:\!\tau_1.e\;x \;\simeq\; e \;:\; (x\!:\!\tau_1) \rightarrow \tau_2$$

Moving on, nothing to see

Université de Montréal

Adding η to iCCω's

$$\Gamma \vdash \lambda x\!:\!\tau_1.e\;x \;\simeq\; e \;:\; (x\!:\!\tau_1) \rightarrow \tau_2$$

For our encoding to preserve types we need:

$$[\Gamma] \vdash [\lambda x\!:\!\tau.e\;x] \;\simeq\; [e] \;:\; [\![(x\!:\!\tau_1) \rightarrow \tau_2]\!]$$

Which expands to:

$$\langle\, |\lambda x\!:\![\![\tau_1]\!].[e\;x]|, \text{iserased}\;((x\!:\![\![\tau_1]\!]) \rightarrow [\![\tau_2]\!])\,\rangle \;\simeq\; \langle [e].0, [e].1\rangle$$

Which then splits into:

$$|\lambda x\!:\![\![\tau_1]\!].\text{bind}\;[e].0\;\lambda f.f\;x| \;\simeq\; [e].0$$
$$\text{iserased}\;((x\!:\![\![\tau_1]\!]) \rightarrow [\![\tau_2]\!]) \;\simeq\; [e].1$$

Encoding iCCω's η into rCCω

Encoding iCCω's η into rCCω requires additional equality rules in rCCω:

Encoding iCCω's η into rCCω

Encoding iCCω's η into rCCω requires additional equality rules in rCCω:

$$\Gamma \vdash \langle e.0, e.1 \rangle \simeq e : \tau_1 \times \tau_2$$

η on pairs? Easy!

Encoding iCCω's η into rCCω

Encoding iCCω's η into rCCω requires additional equality rules in rCCω:

$$\Gamma \vdash \langle e.0, e.1 \rangle \;\simeq\; e \;:\; \tau_1 \times \tau_2$$

η on pairs? Easy!

$$\Gamma \vdash \textsf{iserased}\ \tau \;\simeq\; e \;:\; \textsf{IsErased}\ ||\tau||$$

Definitional proof irrelevance on iserased? ... OK

Encoding iCCω's η into rCCω

Encoding iCCω's η into rCCω requires additional equality rules in rCCω:

$$\Gamma \vdash \langle e.0, e.1 \rangle \simeq e : \tau_1 \times \tau_2$$

η on pairs? Easy!

$$\Gamma \vdash \text{iserased}\ \tau \simeq e : \text{IsErased}\ ||\tau||$$

Definitional proof irrelevance on iserased? ... OK

$$\Gamma \vdash |\lambda x\!:\!\tau_1.\text{bind}\ e\ \lambda f.f\ x| \simeq e : ||(x\!:\!\tau_1) \rightarrow (\tau_2 \times \text{IsErased}\ \tau_2)||$$

Really?

η returns

Mapping rCCω's new η rule on bind back to iCCω:

$$|\lambda x\!:\!\tau_1.\mathsf{bind}\ e\ \lambda f.f\ x| \ \simeq\ e$$

$$\begin{array}{l}\lambda t\!:\!\mathsf{Type}_0.\lambda k\!:\!((x\!:\!\tau_1) \to (\tau_2 \times \mathsf{IsErased}\ \tau_2)) \to t. \\ k\ (\lambda x\!:\!\tau_1.\mathsf{bind}\ e\ \lambda f.f\ x)\end{array} \ \simeq\ e$$

$$k\ (\lambda x\!:\!\tau_1.e\ (\tau_2 \times \mathsf{IsErased}\ \tau_2)\ \lambda f.f\ x) \ \simeq\ e\ t\ k$$

Erasing the “obvious” implicit-able subterms:

$$k\ (\lambda x.e\ \lambda f.f\ x) \ \simeq\ e\ k$$

By parametricity?

Inductive types

Trivial both ways for higher universes (as in UTT)

rCCω to iCCω depends on “impredicative Set” vs “impredicative Prop”

More delicate for iCCω to rCCω where we need to:

- Erase Type_0 inductives

  Gets in the way of strong elimination

- Make sure the encoded types still satisfy positivity

  $||\tau||$ needs to be strictly positive (e.g. not double negation)

- Make sure the encoded terms still satisfy the termination check

Université de Montréal

Problem: Strong Elimination

Can't directly perform strong elimination of erased Type_0 inductives

$$\mathsf{bind}\ [e].0\ \lambda x.\mathsf{Elim}(x, \lambda\vec{x}.[\![e_r\ \vec{x}]\!], [\vec{e}])$$

If e is an (erased) n-tuple, we could reify it with n separate binds

Can't reify an erased dependent tuples or an equality proof

$\Longrightarrow$ Don't erase if not needed!

- Weaken IsErased to match isProp
- Erase large inductives and multi-constructor inductives
- Prove IsErased for single-constructor small inductives

DIRO

Conclusion

Show an equivalence between basic iCCω and rCCω

Doesn't include η

Extends to UTT-style inductive types

Non-identity round-trip

Inductive types in Prop: WiP

Confirm everyone's intuition that Prop and resizing are comparable

Might hopefully help translate proofs between such systems

Next up: resizing impredicative universe polymorphism?