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

\( \sim 1389, \text{ Chaucer: Can’t trust this dude, he’s too impredicative!} \)

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 $\lambda$-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:

\[ \tau_2 : \text{Prop} \implies (x : \tau_1) \rightarrow \tau_2 : \text{Prop} \]

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

Resizing axioms:

\[ \tau : \text{Type}_\ell \wedge P(\tau) \implies \tau : \text{Type}_{\ell'} \]

Most famously, HoTT’s propositional resizing

Impredicative universe polymorphism:

\[ \tau_2 : \text{Type}_\ell \implies (l : \text{Level}) \equiv \tau_2 : \text{Type}_{\ell[0/l]} \]

Foolish experiment in Typer, proposed in last episode
Introduction

The Question

How do forms of impredicativity relate?

⇓

How do impredicative universes compare to resizing axioms?

⇓

How does impredicative Prop compare to HoTT propositional resizing?

⇓

Show equivalence of close(?) relatives to those systems
We start from a predicative PTS with a tower of universes:

\[
\begin{align*}
e, \tau & ::= s \mid x \mid (x: \tau_1) \rightarrow \tau_2 \mid \lambda x: \tau. e \mid e_1 \ e_2 \\
S &= \{ \text{Type}_\ell \mid \ell \in \mathbb{N} \} \\
A &= \{ (\text{Type}_\ell : \text{Type}_{\ell+1}) \mid \ell \in \mathbb{N} \} \\
R &= \{ (\text{Type}_{\ell_1}, \text{Type}_{\ell_2}, \text{Type}_{\ell_1 \sqcup \ell_2}) \mid \ell_1, \ell_2 \in \mathbb{N} \}
\end{align*}
\]
**Impredicative universe: iCCω**

We add impredicative quantification to pCCω’s bottom universe:

\[ e, \tau ::= s \mid x \mid (x : \tau_1) \rightarrow \tau_2 \mid \lambda x : \tau. e \mid e_1 \; e_2 \]

- \( S = \{ \text{Type}_\ell \mid \ell \in \mathbb{N} \} \)
- \( A = \{ (\text{Type}_\ell : \text{Type}_{\ell+1}) \mid \ell \in \mathbb{N} \} \)
- \( R = \{ (\text{Type}_{\ell_1}, \text{Type}_{\ell_2}, \text{Type}_{\ell_1 \sqcup \ell_2}) \mid \ell_1, \ell_2 \in \mathbb{N} \} \)
  \[ \cup \{ (\text{Type}_\ell, \text{Type}_0, \text{Type}_0) \mid \ell \in \mathbb{N} \land \ell > 0 \} \]
Resizing axiom: \( rCC_\omega \)

We start from an erasure monad

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

\[
\begin{align*}
\| \cdot \| : & \text{Type}_\ell \rightarrow \text{Type}_0 & (\forall \ell) \\
\| \cdot \| : & (t : \text{Type}_\ell) \rightarrow t \rightarrow \| t \| & (\forall \ell) \\
\text{bind} : & (t_1 : \text{Type}_{\ell_1}) \rightarrow (t_2 : \text{Type}_{\ell_2}) & (\forall \ell_1, \ell_2) \\
& \rightarrow \| t_1 \| \rightarrow (t_1 \rightarrow \| t_2 \|) \rightarrow \| t_2 \| \\
\Gamma \vdash \text{bind}_{\tau_1, \tau_2} | e_1 | \lambda x : \tau_1 . e_2 \simeq e_2 [e_1/x] : \tau_2
\end{align*}
\]

Similar monad used by Arnaud Spiwack to axiomatize Hurkens's paradox
Warm up: translating \( rCC_\omega \) to \( iCC_\omega \)

Encoding \( rCC_\omega \) into \( iCC_\omega \) is easy. We can just provide definitions:

\[
\begin{align*}
||\tau|| &= (t : \text{Type}_0) \rightarrow (\tau \rightarrow t) \rightarrow t \\
|e|_{\tau} &= \lambda t : \text{Type}_0. \lambda k : (\tau \rightarrow t). k \ e \\
\text{bind}_{\tau_1,\tau_2} \ e_1 \ e_2 &= e_1 ||\tau_2|| \ e_2
\end{align*}
\]

And indeed it satisfies the desired reduction:

\[
\begin{align*}
\text{bind} \ |e_1| \ \lambda x : \tau_1. e_2 \\
\Rightarrow & \quad (\lambda \alpha : \text{Type}_0. \lambda f : (\tau_1 \rightarrow \alpha). f \ e_1) \ \tau_2 \ (\lambda x : \tau_1. e_2) \\
\Rightarrow & \quad e_2[e_1/x]
\end{align*}
\]
Encoding Prop using erasure

We want to encode (written \([ \cdot ]\)) \(iCC\omega\) terms into \(rCC\omega\)

Core problem: \((x : \tau_1) \rightarrow \tau_2\) where \(\tau_1 : Type_{>0}\) and \(\tau_2 : Type_0\)

- \(iCC\omega\) puts them in \(Type_0\)
- \(rCC\omega\) would put them in \(Type_\ell\)

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

\[
[(x : \tau_1) \rightarrow \tau_2] = ||(x : [\tau_1]) \rightarrow [\tau_2]||
\]

Inspired by Coq, we erase all \(Type_0\) terms
Translating iCC$_\omega$ to rCC$_\omega$: first try

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

\[ \Gamma \vdash e : \tau \quad \Rightarrow \quad [\Gamma] \vdash [e] : [\tau] \]

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

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

\[ \Rightarrow \text{Weaken bind so it does not require literally} \ ||\tau|| \]

Instead, require a proof of erasure

\[ \Rightarrow \text{Less of a monad} \]

\[ \Rightarrow \text{Also, we need pairs} \]
**New rCCω axioms**

\[
\text{bind} : (t_1 : \text{Type}_\ell) \rightarrow (t_2 : \text{Type}_0) \rightarrow (\forall \ell) \\
\|t_1\| \rightarrow (t_1 \rightarrow (t_2 \times \text{IsErased } t_2)) \rightarrow (t_2 \times \text{IsErased } t_2)
\]

\[
\text{IsErased} : \text{Type}_0 \rightarrow \text{Type}_0
\]

\[
\text{iserased} : (t : \text{Type}_\ell) \rightarrow \text{IsErased } \|t\| \rightarrow (\forall \ell)
\]

\[
\cdot \times \cdot : \text{Type}_0 \rightarrow \text{Type}_0 \rightarrow \text{Type}_0
\]

\[
\langle \cdot, \cdot \rangle : (t_1 : \text{Type}_0) \rightarrow (t_2 : \text{Type}_0) \rightarrow t_1 \rightarrow t_2 \rightarrow (t_1 \times t_2)
\]

\[
\cdot .0 : (t_1 : \text{Type}_0) \rightarrow (t_2 : \text{Type}_0) \rightarrow (t_1 \times t_2) \rightarrow t_1
\]

\[
\Gamma \vdash \langle e_1, e_2 \rangle . 0 \cong e_1 : \tau_1
\]
New translation of $iCC_\omega$ to $rCC_\omega$

$$\begin{align*}
[\tau] &= \begin{cases} 
[\tau] \times \text{IsErased} \ [\tau] & \text{if } \tau : \text{Type}_0 \\
[\tau] & \text{otherwise}
\end{cases}
\end{align*}$$

$$\begin{align*}
[(x:\tau_1) \to \tau_2] &= \begin{cases} 
\| (x: [\tau_1]) \to [\tau_2] \| & \text{if in the Type}_0 \text{ universe} \\
(x: [\tau_1]) \to [\tau_2] & \text{otherwise}
\end{cases}
\end{align*}$$

$$\begin{align*}
[e_1 e_2] &= \begin{cases} 
\text{bind} \ [e_1].0 \lambda f.f \ [e_2] & \text{if } e_1 \text{ in the Type}_0 \text{ universe} \\
[e_1] \ [e_2] & \text{otherwise}
\end{cases}
\end{align*}$$

$$\begin{align*}
[\lambda x:\tau_1.e] &= \begin{cases} 
\langle \ | \lambda x:[\tau_1].[e] |, \text{isErased} \ ((x: [\tau_1]) \to [\tau_2]) \rangle & \text{if not in the Type}_0 \text{ universe} \\
\lambda x: [\tau_1].[e] & \text{otherwise}
\end{cases}
\end{align*}$$
Translating new $r\text{CC}_\omega$ to $i\text{CC}_\omega$

\[
||\tau|| = (t : \text{Type}_0) \rightarrow (\tau \rightarrow t) \rightarrow t
\]

\[
|e|_\tau = \lambda t : \text{Type}_0.\lambda k : (\tau \rightarrow t).k\ e
\]

\[
\text{bind}_{\tau_1, \tau_2} e_1 e_2 = e_1 (\tau_2 \times \text{IsErased } t_2)\ e_2
\]

\[
\text{IsErased } \tau = (t : \text{Type}_0) \rightarrow t \rightarrow t
\]

\[
\text{iserased } \tau = \lambda t : \text{Type}_0.\lambda x : t.x
\]

\[
\tau_1 \times \tau_2 = \tau_1
\]

\[
\langle e_1, e_2 \rangle = e_1
\]

\[
e.0 = e
\]
Differences to traditional Prop

\(\text{iCC}\omega\) differs from the usual impredicative \(\text{CC}\omega\):

- Both predicative and impredicative functions from \(\text{Type}_\ell\) to \(\text{Type}_0\)
  Used to simplify the rCC\(\omega\) encoding
  Could be replaced by cummulativity of universes
Comparison to propositional resizing

The axioms of \( \text{rCC}_\omega \) are satisfied by HoTT's axioms:

\[
\|\tau\| \leq \text{propositional truncation and resizing}
\]

\[
\text{IsErased} \leq \text{isProp}
\]

\[
\text{bind} \leq \text{the recursion principle of propositional truncation}
\]

They are less general than those of HoTT:

- \( \text{rCC}_\omega \)'s \( \|\tau\| \) does not imply proof irrelevance

- \( \text{IsErased} \) can be fulfilled only by erasure/truncation rather than by a proof of irrelevance
Adding \( \eta \) to rCC\( \omega \)'s

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

Moving on, nothing to see
Adding $\eta$ to iCC$\omega$’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:

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

Which then splits into:

\[ |\lambda x : [\tau_1].\text{bind } [e].0 \ \lambda f . f \ x| \ \simeq \ [e].0 \]

\[ \text{iseraised } ((x : [\tau_1]) \rightarrow [\tau_2]) \ \simeq \ [e].1 \]
Encoding $iCC_\omega$’s $\eta$ into $rCC_\omega$

Encoding $iCC_\omega$’s $\eta$ into $rCC_\omega$ requires additional equality rules in $rCC_\omega$: 
Encoding iCCω’s $\eta$ into rCCω

Encoding iCCω’s $\eta$ into rCCω requires additional equality rules in rCCω:

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

$\eta$ on pairs? Easy!
Encoding $iCC_\omega$’s $\eta$ into $rCC_\omega$

Encoding $iCC_\omega$’s $\eta$ into $rCC_\omega$ requires additional equality rules in $rCC_\omega$:

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

$\eta$ on pairs? Easy!

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

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?
Mapping rCCω’s new \( \eta \) rule on \texttt{bind} back to iCCω:

\[
|\lambda x : \tau_1 . \text{bind } e \lambda f. f \ x| \ \simeq \ e
\]

\[
\lambda t : \text{Type}_0 . \lambda k : ((x : \tau_1) \rightarrow (\tau_2 \times \text{IsErased } \tau_2)) \rightarrow t. \ \simeq \ e
\]

\[
k (\lambda x : \tau_1 . \text{bind } e \lambda f. f \ x)
\]

\[
k (\lambda x : \tau_1 . e (\tau_2 \times \text{IsErased } \tau_2) \lambda f. f \ x) \ \simeq \ e \ t \ k
\]

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)

\( \text{rCC}\omega \) to \( \text{iCC}\omega \) depends on “impredicative Set” vs “impredicative Prop”

More delicate for \( \text{iCC}\omega \) to \( \text{rCC}\omega \) where we need to:

- Erase \( \text{Type}_0 \) 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
Problem: Strong Elimination

Can’t directly perform strong elimination of erased $\texttt{Type}_0$ inductives

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

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

$\implies$ Don’t erase if not needed!

- Weaken $\text{IsErased}$ to match $\text{isProp}$
- Erase large inductives and multi-constructor inductives
- Prove $\text{IsErased}$ for single-constructor small inductives
Show an equivalence between basic $\text{iCC}_\omega$ and $\text{rCC}_\omega$

Doesn’t include $\eta$

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?