

# The case for Impredicative Universe Polymorphism

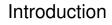
Stefan Monnier

monnier@iro.umontreal.ca

Université de Montréal



Stefan Monnier Resizing Prop 1/17





# 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





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

Typer: low-level  $\lambda$ -calculus intermediate language

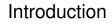
Impredicativity used in:

- 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

Not fond of a special Prop universe (and didn't know about PR)







# Forms of impredicativity

Impredicative universes:  $au_2$ : Prop  $\implies$   $(x: au_1) \rightarrow au_2$ : Prop As present in System F, Coq, Lean, and many others.

Unsound:

Type : Type

Clearly not ideal, especially with erasure.

New, IUP:

The present suggestion

 $\frac{\Gamma, l: \text{Level} \vdash \tau: \text{Type}_u}{\Gamma \vdash (l: \text{Level}) \rightarrow \tau: \text{Type}_{u[0/l]}}$ 





Voices in my head:

- Encoding inductive types as closures.
- Encoding closures as inductive types.

Bounds:

- Strong sums defeat stratification.
- Encode System F

Encouraging signs

• Girard's Paradox did not bite (yet?).





## Encoding inductive types as closures

Church-style encoding of lists:

 $\textit{List}\,\tau \ = \ (t : \mathsf{Type}) \to t \to (\tau \to t \to t) \to t$ 

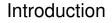
- No induction principle, hence no reasoning.
   Solutions by Awodey et.al. [2018] and Firsov and Stump [2018].
- No strong elimination.

Limited solution by Jenkins et.al. [2021]

Strong elimination via universe polymorphism:

 $\textit{List} \ \tau \ = \ (l: \texttt{Level}) \rightarrow (t: \texttt{Type}_l) \rightarrow t \rightarrow (\tau \rightarrow t \rightarrow t) \rightarrow t$ 







 $(t:\mathsf{Type}_l) \to t \to (\tau \to t \to t) \to t : \mathsf{Type}_{u \sqcup S l}$  $(l:\mathsf{Level}) \to (t:\mathsf{Type}_l) \to t \to (\tau \to t \to t) \to t : \mathsf{Type}_{??}$ 

Predicative principles stipulate  $\sup_{l} (u \sqcup S l)$ :

$$(l: \mathsf{Level}) \to (t: \mathsf{Type}_l) \to \dots : \mathsf{Type}_\omega$$

Yet! The type is isomorphic to the inductive:  $List \tau$  :  $Type_u$ IUP uses  $inf_l (u \sqcup S l)$ :  $(l : Level) \rightarrow (t : Type_l) \rightarrow \dots$  :  $Type_{(u \sqcup 1)}$ 





## Encoding closures as inductive types

Closure conversion turns (open) functions into pairs of: captured environment  $\times$  closed function

 $\begin{array}{l|l} \lambda n.n + 1: \operatorname{Int} \to \operatorname{Int} & \lambda n.n + \operatorname{length} \operatorname{Float} \operatorname{temps}: \operatorname{Int} \to \operatorname{Int} \\ & \Longrightarrow & & \Rightarrow \\ ((), \lambda(\operatorname{env}, n).n + 1) & ((\operatorname{length}, \operatorname{Float}, \operatorname{temps}), \\ & \lambda(\operatorname{env}, n).n + \operatorname{env}.1 \ \operatorname{env}.2 \ \operatorname{env}.3) \end{array}$ 

Hide the type of *env* to preserve types:

 $\mathsf{Int} \to \mathsf{Int} \implies \exists t.(t \times ((t \times \mathsf{Int}) \to \mathsf{Int}))$ 

Works great in System F!

(after erasing *Float*)





#### **Closure conversion with universes**

With universes this turns into:  $\exists (t: \mathsf{Type}_u) . (t \times ((t \times \mathsf{Int}) \to \mathsf{Int}))$ And we need to hide u which depends on the captured environment:

 $\exists (l: \mathsf{Level}). \exists (t: \mathsf{Type}_l). (t \times ((t \times \mathsf{Int}) \to \mathsf{Int}))$ 

Predicative principles stipulate  $\sup_{l} ((S \ l) \sqcup 0)$ :

 $\exists (l: \mathsf{Level}). \exists (t: \mathsf{Type}_l). \ldots : \mathsf{Type}_{\omega}$ 

Yet! The type is equivalent to the arrow type: Int  $\rightarrow$  Int : Type<sub>0</sub> IUP uses  $\inf_{l} ((S l) \sqcup 0)$ :  $\exists (l : Level) . \exists (t : Type_{l}) ... : Type_{1}$ 





## Strong sums

Let's try IUP with strong sums:

We can define:

 $\frac{\Gamma, l: \mathsf{Level} \vdash \tau: \mathsf{Type}_u}{\Gamma \vdash \Sigma l. \tau: \mathsf{Type}_{u[0/l]}}$ 

 $\begin{aligned} &\textit{lower}\left(l:\text{Level}\right)\left(t:\text{Type}_l\right)\left(x:t\right) = \langle l, \langle t, x \rangle \rangle \\ &\textit{raise}\left(b:\Sigma l.\Sigma t:\text{Type}_l.t\right) = b.2.2 \end{aligned}$ 

This gives us:

 $lower u \tau x : \Sigma l.\Sigma t: \mathsf{Type}_l.t : \mathsf{Type}_1$  $\forall x:\tau: \mathsf{Type}_u. \quad \textit{raise} (lower u \tau x) \rightsquigarrow x$ 

We can smuggle any value in a box that lives in Type<sub>1</sub>!

Suggests that IUP is incompatible with first-class universe levels.





### System F

IUP is as strong as System F:

$$\begin{split} \llbracket \forall t.\tau \rrbracket &= (l: \mathsf{Level}) \to (t: \mathsf{Type}_l) \to \llbracket \tau \rrbracket \\ \llbracket \Lambda t.e \rrbracket &= \lambda(l: \mathsf{Level}).\lambda(t: \mathsf{Type}_l).\llbracket e \rrbracket \\ \llbracket e[\tau] \rrbracket &= \llbracket e \rrbracket \; u \; \llbracket \tau \rrbracket \end{split}$$

We can just compute u from  $\llbracket au 
rbracket$ 





Common example of inconsistency in impredicative systems:

```
\begin{aligned} \textit{Ordering}: \mathsf{Type} &= \Sigma(\textit{set}: \mathsf{Type}_u). \\ & \Sigma(\textit{less-than}:\textit{set} \to \textit{set} \to \textit{Type}). \end{aligned}
```

The inconsistency appears when we define an ordering of orderings.

In a predicative setting this does not work because *Ordering* ends up in a universe level higher than u.

If we want to try and reproduce the paradox using IUP, we need to abstract over the universe level of *set*.





# Well ordering via existential quantification

First attempt:

 $\begin{aligned} \textit{Ordering1}: \mathsf{Type}_1 &= \exists (l: \mathsf{Level}). \\ & \Sigma(\textit{set}: \mathsf{Type}_l). \\ & \Sigma(\textit{less-than}: \textit{set} \to \textit{set} \to \mathsf{Type}_0). \end{aligned}$ 

We can now instantiate set to this type.

But the weak nature of the existential makes *Ordering1* unusable: We cannot eliminate to anything that depends on l, so ... We cannot eliminate to anything that depends on *set*, so ...

. . .



Stefan Monnier Resizing Prop 13/17



# Well ordering via universal quantification

Second attempt:

 $\begin{array}{l} \textit{Ordering2}: \texttt{Type}_1 = (l:\texttt{Level}) \rightarrow \\ & \Sigma(\textit{set}:\texttt{Type}_l). \\ & \Sigma(\textit{less-than}:\textit{set} \rightarrow \textit{set} \rightarrow \texttt{Type}_0). \end{array}$ 

Again, we can now instantiate set to this type (when l is 1).

. . .

¿Write a function which instantiates *set* to *Ordering2* when l is 1 yet to something in Type<sub>0</sub> when l is 0?

Use set : Type<sub>S l</sub> to avoid the Type<sub>0</sub> case? Pushes Ordering2 to Type<sub>2</sub>!





#### Conclusion

Church-encoding suggests: $(l: Level) \rightarrow \tau : Type_{u[0/l]}$ Closure conversion suggests: $\exists (l: Level).\tau : Type_{u[0/l]}$ Better stop before  $\Sigma(l: Level).\tau : Type_{u[0/l]}$ !IUP is as strong as System F.We have failed to encode known paradoxes so far.We have used only  $(l: Level) \rightarrow (t: Type_l) \rightarrow ...$  so far

¡Help!

