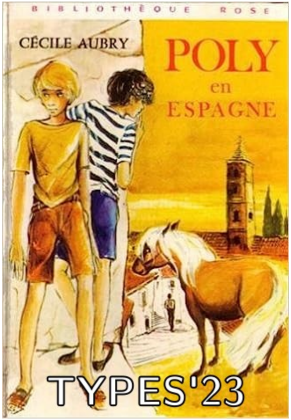
From Lost to the River: Embracing Sort Proliferation

Gaëtan Gilbert, **Pierre-Marie Pédrot**, Matthieu Sozeau, Nicolas Tabareau

INRIA

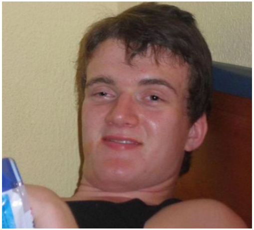


Types, types, types

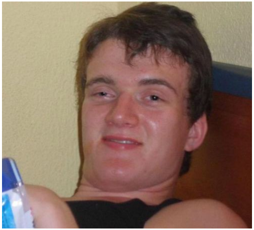
Type theory is about types!

Types, types, types

Type theory is about types!



Type theory is about types!



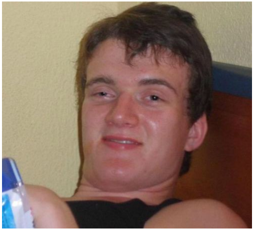
It's all about assigning types to terms.

In MLTT and its variants, types are **also** terms.

So you need also need to give types to types!

Types, types, types

Type theory is about types!



It's all about assigning types to terms.

In MLTT and its variants, types are **also** terms.

So you need also need to give types to types!

The type of a type is a universe.

$$\vdash A : \mathtt{Type} \qquad \sim \qquad A \text{ is a type}$$

Quis custodiet ipsos custodes?

What is the type of the universe?

Quis custodiet ipsos custodes?

What is the type of the universe?

Martin-Löf '71

$$\vdash \texttt{Type} : \texttt{Type}$$

Quis custodiet ipsos custodes?

What is the type of the universe?

Martin-Löf '71

$$\vdash \texttt{Type} : \texttt{Type}$$

Girard, '71 and 15 minutes

MLTT with the above rule is inconsistent.

What is the type of the universe?

Martin-Löf '71

$$\vdash \mathtt{Type} : \mathtt{Type}$$

Girard, '71 and 15 minutes

MLTT with the above rule is inconsistent.

Standard solution

$$\vdash \mathtt{Type}_n : \mathtt{Type}_{n+1} \qquad \text{for } n \in \mathbb{N}$$

Meanwhile in the Real World™

This might look innocuous in theory, but it's a pain in practice.

Meanwhile in the Real World$^{\text{TM}}$

This might look innocuous in theory, but it's a pain in practice.

- You have to explicitly set a level n for every single `Type`
- If you need one intermediate universe in the middle of a proof...

  $\rightsquigarrow$ Better have used BASIC line numbering!
- Martin-Löf forbid that you want to use a term at two different levels

  $\rightsquigarrow$ Twice as much pleasure of code writing!

Meanwhile in the Real World™

This might look innocuous in theory, but it's a pain in practice.

- You have to explicitly set a level n for every single `Type`
- If you need one intermediate universe in the middle of a proof...

  $\rightsquigarrow$ Better have used BASIC line numbering!
- Martin-Löf forbid that you want to use a term at two different levels

  $\rightsquigarrow$ Twice as much pleasure of code writing!

It's a very **anti-modular** feature!

Well-know issues imply well-known solutions

Thankfully, it's a well-known issue

1. Floating universes aka (global) algebraic constraints

$$\mathfrak{S} \vDash i < j$$

2. Some flavour of universe polymorphism aka bound levels

$$\vdash M : \tilde{\forall} i j.\ \mathtt{Type}_i \to \mathtt{Type}_{j+1}$$

3. More exotic stuff: TEMPLATE POLY, crude but effective...

Well-know issues imply well-known solutions

Thankfully, it's a well-known issue

1. Floating universes aka (global) algebraic constraints

$$\mathfrak{S} \vDash i < j$$

2. Some flavour of universe polymorphism aka bound levels

$$\vdash M : \tilde{\forall} i j.\ \mathtt{Type}_i \to \mathtt{Type}_{j+1}$$

3. More exotic stuff: TEMPLATE POLY, crude but effective...

Not mutually exclusive! Coq uses 1 + 2 + TEMPLATE POLY.

Unexpected Prequel

Modularity is restored, the Galaxy is at peace...

Unexpected Prequel

Modularity is restored, the Galaxy is at peace...

BREAKING NEWS

Nope! There are alternate universes out there.

Rising from the CIC tradition, we had `Prop` for decades in Coq.

Propping up the Scene

Prop: a mishmash of features

- Impredicative: $\Pi x : A.\, B : \texttt{Prop}$ as long as $B : \texttt{Prop}$
- 100% compatible with proof-irrelevance (but not irrelevant)
- Erasable through extraction

Propping up the Scene

Prop: a mishmash of features

- Impredicative: $\Pi x : A.\, B : \texttt{Prop}$ as long as $B : \texttt{Prop}$
- 100% compatible with proof-irrelevance (but not irrelevant)
- Erasable through extraction

Due to these features, elimination of Prop inductives is tricky

- Prop to Prop is fine
- Prop to Type must satisfy **singleton elimination**

Propping up the Scene

`Prop`: a mishmash of features

- Impredicative: $\Pi x : A.\, B : \texttt{Prop}$ as long as $B : \texttt{Prop}$
- 100% compatible with proof-irrelevance (but not irrelevant)
- Erasable through extraction

Due to these features, elimination of `Prop` inductives is tricky

- `Prop` to `Prop` is fine
- `Prop` to `Type` must satisfy **singleton elimination**

This prevents (naive) universe polymorphism over `Prop`

$$\texttt{Inductive Box (A : Type}_i\texttt{) : Type}_j \texttt{ := box : A} \rightarrow \texttt{Box A}$$

It's only the beginning

We have to duplicate everything between `Prop` and `Type`

- All stdlib basic inductives come in two flavours (e.g. $\exists$ vs. Σ)
- Mitigated by `Prop` $\subseteq$ `Type` $\quad\rightsquigarrow\quad$ only the return sort matters
- Weird unification artifacts still

It's only the beginning

We have to duplicate everything between `Prop` and `Type`

- All stdlib basic inductives come in two flavours (e.g. $\exists$ vs. Σ)
- Mitigated by `Prop` $\subseteq$ `Type` $\quad\rightsquigarrow\quad$ only the return sort matters
- Weird unification artifacts still

A new opponent has appeared

... but things went **really south** since the introduction of `SProp`

- Now we need at least three variants
- ... but `SProp` $\not\subseteq$ `Type` $\quad\rightsquigarrow\quad$ 2^n variants for n parameters
- Unification not only weird, but plain unsound

It's only the beginning

We have to duplicate everything between `Prop` and `Type`

- All stdlib basic inductives come in two flavours (e.g. $\exists$ vs. Σ)
- Mitigated by `Prop` $\subseteq$ `Type` $\rightsquigarrow$ only the return sort matters
- Weird unification artifacts still

A new opponent has appeared

... but things went **really south** since the introduction of `SProp`

- Now we need at least three variants
- ... but `SProp` $\not\subseteq$ `Type` $\rightsquigarrow$ 2^n variants for n parameters
- Unification not only weird, but plain unsound

So long for modularity!

What now?

There is no reason to stop at three hierarchies.

- Fibrant vs. strict universes (HoTT)
- Pure vs. impure universes (Exceptional Theory, MTT, ...)
- Setoid / parametric / cubic / blue-haired ω-potatoid universes

What now?

There is no reason to stop at three hierarchies.

- Fibrant vs. strict universes (HoTT)
- Pure vs. impure universes (Exceptional Theory, MTT, ...)
- Setoid / parametric / cubic / blue-haired ω-potatoid universes

Resistance is futile. Let's embrace sort proliferation!

with the dark powers of sort polymorphism

All universes in a single proof assistant

Land of the Free

It's a Revolution

Variables!

Land of the Free

It's a Revolution

Variables!

$$
\begin{array}{lcll}
q & ::= & \alpha \mid \mathbf{Type} \mid \mathbf{Prop} \mid \mathbf{SProp} \mid \ldots & \text{(sort qualities)} \\
M & ::= & \mathtt{Sort}_{\ell}^{q} \quad (\ell \text{ level}, q \text{ quality}) \mid \ldots & \text{(terms)}
\end{array}
$$

- No algebraic structure on qualities
- Usual universes become notations

$$
\mathtt{Type}_{\ell} := \mathtt{Sort}_{\ell}^{\mathbf{Type}} \qquad \mathtt{Prop} := \mathtt{Sort}_{0}^{\mathbf{Prop}} \qquad \mathtt{SProp} := \mathtt{Sort}_{0}^{\mathbf{SProp}}
$$

Give me the Rules

$$\frac{}{\vdash \mathtt{Sort}^{q}_{\ell} : \mathtt{Type}_{\ell+1}}$$

$$\frac{}{\vdash \mathtt{Sort}^{\mathbf{Prop}}_{i} \equiv \mathtt{Sort}^{\mathbf{Prop}}_{j}} \qquad \frac{}{\vdash \mathtt{Sort}^{\mathbf{SProp}}_{i} \equiv \mathtt{Sort}^{\mathbf{SProp}}_{j}}$$

$$\frac{\vdash A : \mathtt{Sort}^{q_A}_{\ell_A} \qquad x : A \vdash B : \mathtt{Sort}^{q_B}_{\ell_B}}{\vdash \Pi x : A.\, B : \mathtt{Sort}^{q_B}_{\ell_A \vee \ell_B}}$$

- `Type` always classifies sorts
- Impredicativity implemented as level-irrelevance
- Product rule is call-by-name-ish / impredicative-friendly

Give me the Rules

$$\frac{}{\vdash \mathtt{Sort}^{q}_{\ell} : \mathtt{Type}_{\ell+1}}$$

$$\frac{}{\vdash \mathtt{Sort}^{\mathbf{Prop}}_{i} \equiv \mathtt{Sort}^{\mathbf{Prop}}_{j}} \qquad \frac{}{\vdash \mathtt{Sort}^{\mathbf{SProp}}_{i} \equiv \mathtt{Sort}^{\mathbf{SProp}}_{j}}$$

$$\frac{\vdash A : \mathtt{Sort}^{q_A}_{\ell_A} \qquad x : A \vdash B : \mathtt{Sort}^{q_B}_{\ell_B}}{\vdash \Pi x : A.\, B : \mathtt{Sort}^{q_B}_{\ell_A \vee \ell_B}}$$

- Type always classifies sorts
- Impredicativity implemented as level-irrelevance
- Product rule is call-by-name-ish / impredicative-friendly

These rules are compatible with all our intended instances

Qot qot quantum

We extend Coq universe polymorphism with sort polymorphism.

$$\mathbf{c} : \forall(\alpha_1 \ldots \alpha_n \in q).\, \forall(i_1 \ldots i_m \in \ell \mid \mathfrak{S}).\, A$$

Qot qot quantum

We extend Coq universe polymorphism with sort polymorphism.

$$\mathbf{c} : \forall(\alpha_1 \ldots \alpha_n \in q).\, \forall(i_1 \ldots i_m \in \ell \mid \mathfrak{S}).\, A$$

$$\frac{\mathbf{c} : \forall(\alpha_1 \ldots \alpha_n \in q).\, \forall(i_1 \ldots i_m \in \ell \mid \mathfrak{S}).\, A \qquad \mathfrak{S}_0 \vDash \mathfrak{S}\{\vec{i} := \vec{\ell}\}}{\mathfrak{S}_0 \mid \Gamma \vdash \mathbf{c}\{q_1 \ldots q_n \mid \ell_1 \ldots \ell_m\} : A\{\vec{\alpha} := \vec{q}, \vec{i} := \vec{\ell}\}}$$

- Prenex, external polymorphism
- Generalization of universe polymorphism, compatible with it
- No constraint system on sorts! (†)

Don't be so Negative

What about inductive types?

Don't be so Negative

What about inductive types?

We have a scheme that is compatible with non-sort polymorphic code.

- Introduction rules for inductive types are unchanged.
- The tricky part is elimination: we generalize singleton criteria

$M : \mathcal{I} : \mathtt{Sort}^{q}$ and $T : \mathtt{Sort}^{r} \vdash \mathtt{case}\ M\ \mathtt{return}\ T\ \mathtt{with}\ \ldots$ allowed?

Don't be so Negative

What about inductive types?

We have a scheme that is compatible with non-sort polymorphic code.

- Introduction rules for inductive types are unchanged.
- The tricky part is elimination: we generalize singleton criteria

$M : \mathcal{I} : \mathtt{Sort}^{q}$ and $T : \mathtt{Sort}^{r} \vdash \mathtt{case}\ M\ \mathtt{return}\ T\ \mathtt{with}\ \ldots$ allowed?

q	r
α	$\{\alpha\}$
Type	any
Prop	any if $\mathcal{I}$ singleton (finite quality check) $\{\mathtt{Prop}, \mathtt{SProp}\}$ otherwise
SProp	any if $\mathcal{I}$ empty $\{\mathtt{SProp}\}$ otherwise

Final product

TYPE THEORY IS AT PEACE.

Final product

TYPE THEORY IS AT PEACE.

- Sort-poly is a conservative extension (like univ-poly)
- ... just a glorified copy-paste!
- In particular, it doesn't change the consistency of the ambient theory
- It blends easily with univ-poly

Proof.

All typing rules are stable by substitution with ground qualities. □

Unexpected Byproduct

TEMPLATE POLY

- a primitive attempt at univ-poly
- a neverending stream of `False`
- Hard to specify and thus not well-understood

Unexpected Byproduct

TEMPLATE POLY

- a primitive attempt at univ-poly
- a neverending stream of `False`
- Hard to specify and thus not well-understood

We can now model it with cumulative inductives + sort-poly!

Template-poly inductives are:

- polymorphic cumulative types whose universe levels are all irrelevant
- levels must satisfy a syntactic “inferability” criterion
- template parameters are sort-poly with the same sort as inductive

TL;DR: TEMPLATE POLY has an intended semantics (†)

What is to be done

The sort-poly infrastructure is now part of Coq **unification** (8.17).

- Part of the kernel term representation
- Used to delay sort assignment in universe unification
- Dedicated handling of $\texttt{Prop} \subseteq \texttt{Sort}^q \subseteq \texttt{Type}$
- Solves longstanding plaguing issues with `SProp` and `Prop`

What is to be done

The sort-poly infrastructure is now part of Coq **unification** (8.17).

- Part of the kernel term representation
- Used to delay sort assignment in universe unification
- Dedicated handling of $\texttt{Prop} \subseteq \texttt{Sort}^q \subseteq \texttt{Type}$
- Solves longstanding plaguing issues with `SProp` and `Prop`

No corresponding typing rules in **kernel**.

- No quantification on constants yet
- Kernel fails on unbound sort qualities (like evars)
- In the process of being implemented

What is to be done

The sort-poly infrastructure is now part of Coq **unification** (8.17).

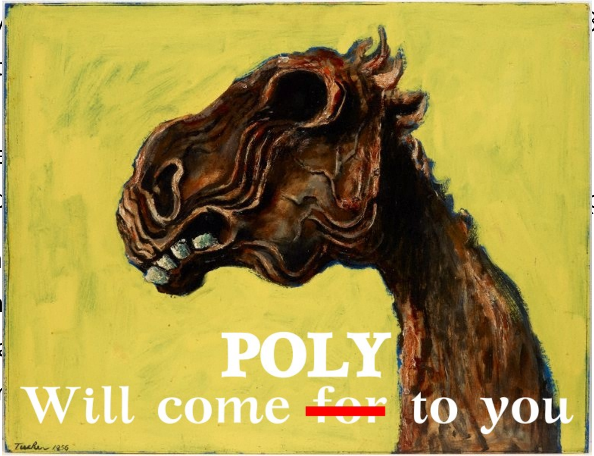
- Part of the kernel term representation
- Used to delay sort assignment in universe unification
- Dedicated handling of $\texttt{Prop} \subseteq \texttt{Sort}^q \subseteq \texttt{Type}$
- Solves longstanding plaguing issues with `SProp` and `Prop`

No corresponding typing rules in **kernel**.

- No quantification on constants yet
- Kernel fails on unbound sort qualities (like evars)
- In the process of being implemented

THE TIME FOR MULTIVERSE EXPANSION HAS COME.

What is to be done

The sort-poly  .17).

- Part of t
- Used to
- Dedicate
- Solves lo

No correspon

- No quan
- Kernel f
- In the pr

THE TIME FOR MULTIVERSE EXPANSION HAS COME.

Scribitur ad narrandum, non ad probandum

Thanks for your attention.