Bounded Sort Polymorphism with Elimination Constraints

POPL 2026, Rennes, France

Johann Rosain[1] Tomás Díaz[2] Kenji Maillard[3] Matthieu Sozeau[3] Nicolas Tabareau[3] Éric Tanter[2] Théo Winterhalter[4]

January 16, 2026

[1]École Normale Supérieure de Lyon, Lyon, France

[2]PLEIAD Lab, Computer Science Department (DCC), University of Chile, Chile

[3]Galinette Project Team, LS2N & Inria de l'Université de Rennes, Nantes, France

[4]LMF & Inria Saclay, Saclay, France

A Bit of a Dry Start, Right?

A Bit of a Dry Start, Right?

Elder's Wisdom

> *'Avoid requiring something to be stated more than once; factor out the recurring pattern.'*
>
> — Bruce J. MacLennan (1999)

A Bit of a Dry Start, Right?

Elder's Wisdom

'Avoid requiring something to be stated more than once; factor out the recurring pattern.' — Bruce J. MacLennan (1999)

'Two or more? use a for.' — Edsger Dijkstra (probably between 1952 and 1999)

A Bit of a Dry Start, Right?

Elder's Wisdom

> *'Avoid requiring something to be stated more than once; factor out the recurring pattern.'* — Bruce J. MacLennan (1999)
>
> *'Two or more? use a for.'* — Edsger Dijkstra (probably between 1952 and 1999)

Programming languages facilitated this principle for a while:

- higher-order functions,
- polymorphism,
- etc.

A Bit of a Dry Start, Right?

Elder's Wisdom

> *'Avoid requiring something to be stated more than once; factor out the recurring pattern.'* — Bruce J. MacLennan (1999)
>
> *'Two or more? use a for.'* — Edsger Dijkstra (probably between 1952 and 1999)

Programming languages facilitated this principle for a while:

- higher-order functions,
- polymorphism,
- etc.

That was before multi-sorted type theories *join the battle*.

Multi-Sorted What?

Setting:

$$t, u \quad ::= \quad x \mid t\ u \mid \lambda x.\ t \mid \Pi(x : t).\ u \mid \texttt{Type}_\ell$$

+ inductives and records (and their introduction/elimination).

where

- x is a variable
- ℓ is a universe level

Multi-Sorted What?

Setting:

$$t, u \quad ::= \quad x \mid t\ u \mid \lambda x.\ t \mid \Pi(x : t).\ u \mid \cancel{\mathsf{Type}_\ell} \mid \mathcal{U}_\ell^s$$

+ inductives and records (and their introduction/elimination).

where

- x is a variable
- ℓ is a universe level
- s is a *sort* in $\mathbb{G}$

Multi-Sorted What?

Setting:

$$t, u \quad ::= \quad x \mid t\ u \mid \lambda x.\ t \mid \Pi(x : t).\ u \mid \cancel{\mathtt{Type}_\ell} \mid \mathcal{U}_\ell^s$$

+ inductives and records (and their introduction/elimination).

where

- x is a variable
- ℓ is a universe level
- s is a *sort* in $\mathbb{G}$

in $\mathbb{G}$, we have

- a distinguished sort $\mathtt{Type}$ s.t. $\vdash \mathcal{U}_\ell^s : \mathcal{U}_{\ell+1}^{\mathtt{Type}}$,
- predicative sorts $\mathbb{G}_P$,
- impredicative sorts $\mathbb{G}_I$.

Multi-Sorted What?

Setting:

$$t, u \quad ::= \quad x \mid t\ u \mid \lambda x.\ t \mid \Pi(x : t).\ u \mid \cancel{\mathtt{Type}_\ell} \mid \mathcal{U}_\ell^s$$

+ inductives and records (and their introduction/elimination).

where

- x is a variable
- ℓ is a universe level
- s is a *sort* in $\mathbb{G}$

in $\mathbb{G}$, we have

- a distinguished sort $\mathtt{Type}$ s.t. $\vdash \mathcal{U}_\ell^s : \mathcal{U}_{\ell+1}^{\mathtt{Type}}$,
- predicative sorts $\mathbb{G}_P$,
- impredicative sorts $\mathbb{G}_I$.

By s_I *impredicative*, we mean:

$$\frac{\Gamma \vdash A : \mathcal{U}_\ell^s \qquad \Gamma, x : A \vdash B : \mathcal{U}_{\ell'}^{s_I}}{\Gamma \vdash \Pi(x : A).\ B : \mathcal{U}_{\ell'}^{s_I}}$$

Grounded in Reality

Ground sorts:

- Agda: $\mathbb{G}_P = \{\texttt{Set}, \texttt{Prop}\}$, $\mathbb{G}_I = \emptyset$,
- Lean: $\mathbb{G}_P = \{\texttt{Type}\}$, $\mathbb{G}_I = \{\texttt{Prop}\}$,
- Rocq: $\mathbb{G}_P = \{\texttt{Type}\}$, $\mathbb{G}_I = \{\texttt{Prop}, \texttt{SProp}\}$.

Grounded in Reality

Ground sorts:

- Agda: $\mathbb{G}_P = \{\texttt{Set}, \texttt{Prop}\}$, $\mathbb{G}_I = \emptyset$,
- Lean: $\mathbb{G}_P = \{\texttt{Type}\}$, $\mathbb{G}_I = \{\texttt{Prop}\}$,
- Rocq: $\mathbb{G}_P = \{\texttt{Type}\}$, $\mathbb{G}_I = \{\texttt{Prop}, \texttt{SProp}\}$.

Sort	Attributes
`Type`/`Set`	computational content (booleans, integers, etc), types
`Prop`/`SProp`	(irrelevant) proofs

Grounded in Reality

Ground sorts:

- Agda: $\mathbb{G}_P = \{\texttt{Set}, \texttt{Prop}\}$, $\mathbb{G}_I = \emptyset$,
- Lean: $\mathbb{G}_P = \{\texttt{Type}\}$, $\mathbb{G}_I = \{\texttt{Prop}\}$,
- Rocq: $\mathbb{G}_P = \{\texttt{Type}\}$, $\mathbb{G}_I = \{\texttt{Prop}, \texttt{SProp}\}$.

Sort	Attributes
`Type`/`Set`	computational content (booleans, integers, etc), types
`Prop`/`SProp`	(irrelevant) proofs

Induce a small restriction: cannot create computational content from proofs.

Wet Toes in this Dry Afternoon

In practice,[1] also leads to the *dreaded* duplication:

[1] Disclaimer: all examples are in Rocq's syntax.

Wet Toes in this Dry Afternoon

In practice,[1] also leads to the *dreaded* duplication:

```
Inductive sum (A B : Type) : Type :=
| inl : A → sum A B
| inr : B → sum A B.
```

[1] Disclaimer: all examples are in Rocq's syntax.

Wet Toes in this Dry Afternoon

In practice,[1] also leads to the *dreaded* duplication:

```
Inductive sum (A B : Type) : Type :=
| inl : A → sum A B
| inr : B → sum A B.

Inductive or (A B : Prop) : Prop :=
| or_introl : A → or A B
| or_intror : B → or A B.
```

[1] Disclaimer: all examples are in Rocq's syntax.

Wet Toes in this Dry Afternoon

In practice,[1] also leads to the *dreaded* duplication:

```
Inductive sum (A B : Type) : Type :=
| inl : A → sum A B
| inr : B → sum A B.

Inductive sumbool (A B : Prop) : Type :=
| left : A → sumbool A B
| right : B → sumbool A B.

Inductive or (A B : Prop) : Prop :=
| or_introl : A → or A B
| or_intror : B → or A B.
```

[1] Disclaimer: all examples are in Rocq's syntax.

Not All Heroes Wear Capes

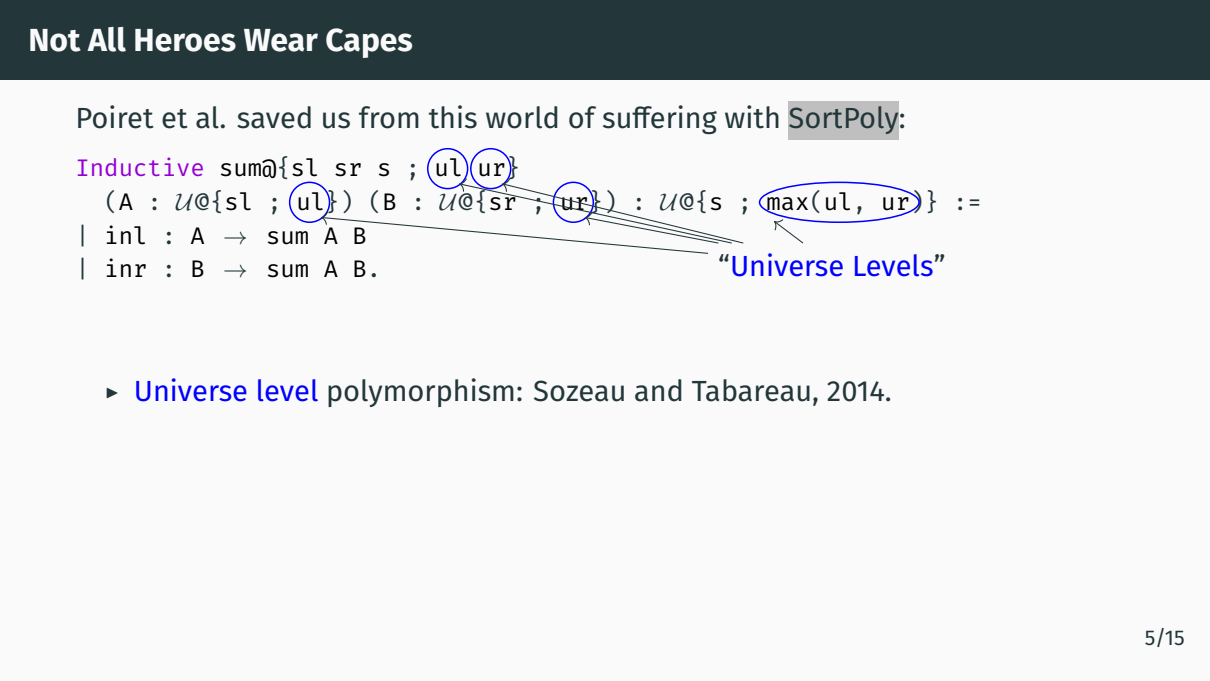
Poiret et al. saved us from this world of suffering with SortPoly:

```
Inductive sum@{sl sr s ; ul ur}
  (A : 𝒰@{sl ; ul}) (B : 𝒰@{sr ; ur}) : 𝒰@{s ; max(ul, ur)} :=
| inl : A → sum A B
| inr : B → sum A B.
```

Not All Heroes Wear Capes

Poiret et al. saved us from this world of suffering with SortPoly:

```
Inductive sum@{sl sr s ; ul ur}
  (A : 𝒰@{sl ; ul}) (B : 𝒰@{sr ; ur}) : 𝒰@{s ; max(ul, ur)} :=
| inl : A → sum A B
| inr : B → sum A B.
```



- Universe level polymorphism: Sozeau and Tabareau, 2014.

Not All Heroes Wear Capes

Poiret et al. saved us from this world of suffering with SortPoly:

```
Inductive sum@{sl sr s ; ul ur}
  (A : 𝒰@{sl ; ul}) (B : 𝒰@{sr ; ur}) : 𝒰@{s ; max(ul, ur)} :=
| inl : A → sum A B
| inr : B → sum A B.
```

- Universe level polymorphism: Sozeau and Tabareau, 2014.
- In 2025, Poiret et al. bring *prenex* sort polymorphism.

Not All Heroes Wear Capes

Poiret et al. saved us from this world of suffering with SortPoly:

```
Inductive sum@{sl sr s ; ul ur}
  (A : 𝒰@{sl ; ul}) (B : 𝒰@{sr ; ur}) : 𝒰@{s ; max(ul, ur)} :=
| inl : A → sum A B
| inr : B → sum A B.
```


- Universe level polymorphism: Sozeau and Tabareau, 2014.
- In 2025, Poiret et al. bring *prenex* sort polymorphism.
- We now have full universe polymorphism.

Not All Heroes Wear Capes

Poiret et al. saved us from this world of suffering with SortPoly:

```
Inductive sum@{sl sr s ; ul ur}
  (A : 𝒰@{sl ; ul}) (B : 𝒰@{sr ; ur}) : 𝒰@{s ; max(ul, ur)} :=
| inl : A → sum A B
| inr : B → sum A B.
```

- Universe level polymorphism: Sozeau and Tabareau, 2014.
- In 2025, Poiret et al. bring *prenex* sort polymorphism.
- We now have full universe polymorphism.

But this is not enough to avoid duplication!

A Non-Improved Situation

With unbounded sort polymorphism: cannot define *e.g.*, a generic eliminator:

```
Definition sum_elim@{sl sr s s'}
  {A : 𝒰@{sl}} {B : 𝒰@{sr}} {C : sum@{sl sr s} A B → 𝒰@{s'}}
  (u : sum@{sl sr s} A B) (f : forall (x : A), C (inl x))
  (g : forall (y : B), C (inr y)) : C u :=
  match u with
  | inl a ⇒ f a
  | inr b ⇒ g b
  end.
```

A Non-Improved Situation

With unbounded sort polymorphism: cannot define *e.g.*, a generic eliminator:

```
Definition sum_elim@{sl sr Prop Type}
  {A : 𝒰@{sl}} {B : 𝒰@{sr}} {C : sum@{sl sr Prop} A B → 𝒰@{Type}}
  (u : sum@{sl sr Prop} A B) (f : forall (x : A), C (inl x))
  (g : forall (y : B), C (inr y)) : C u :=
  match u with
  | inl a ⇒ f a
  | inr b ⇒ g b
  end.
```

Otherwise, setting $s := \texttt{Prop}$ and $s' := \texttt{Type}$ makes the type theory inconsistent (e.g., Hurkens).

A Non-Improved Situation

With unbounded sort polymorphism: cannot define *e.g.*, a generic eliminator:

```
Definition sum_elim@{sl sr Prop Type}
  {A : 𝒰@{sl}} {B : 𝒰@{sr}} {C : sum@{sl sr Prop} A B → 𝒰@{Type}}
  (u : sum@{sl sr Prop} A B) (f : forall (x : A), C (inl x))
  (g : forall (y : B), C (inr y)) : C u :=
  match u with
  | inl a ⇒ f a
  | inr b ⇒ g b
  end.
```

Otherwise, setting $s := \texttt{Prop}$ and $s' := \texttt{Type}$ makes the type theory inconsistent (e.g., Hurkens). It is also the case when setting $s := \texttt{SProp}$.

A Non-Improved Situation

With unbounded sort polymorphism: cannot define *e.g.*, a generic eliminator:

```
Definition sum_elim@{sl sr Prop Type}
  {A : 𝒰@{sl}} {B : 𝒰@{sr}} {C : sum@{sl sr Prop} A B → 𝒰@{Type}}
  (u : sum@{sl sr Prop} A B) (f : forall (x : A), C (inl x))
  (g : forall (y : B), C (inr y)) : C u :=
  match u with
  | inl a ⇒ f a
  | inr b ⇒ g b
  end.
```

Otherwise, setting $s := \texttt{Prop}$ and $s' := \texttt{Type}$ makes the type theory inconsistent (e.g., Hurkens). It is also the case when setting $s := \texttt{SProp}$.

Still need to declare the different eliminators “by hand”.

Contributions

In SortPoly, no principal sort assignment for e.g.:

```
Fixpoint map {A B} (f: A → B) (l: list A): list B :=
  match l with [] ⇒ [] | a :: t ⇒ f a :: map f t end.
```

Contributions

In SortPoly, no principal sort assignment for e.g.:

```
Fixpoint map {A B} (f: A → B) (l: list A): list B :=
  match l with [] ⇒ [] | a :: t ⇒ f a :: map f t end.
```

We introduce a bounded sort-polymorphic type system such that it:

- allows writing the most generic terms (w.r.t. sorts),
- allows infering the principal sort assignment,
- has relative consistency (w.r.t. SortPoly),
- has the *same* logical power as SortPoly.

The Missing Ingredient

Our proposal: interactions between sorts using *elimination constraints*.

```
Definition sum_elim@{sl sr s s'|s ⇝ s'}
```

The Missing Ingredient

Our proposal: interactions between sorts using *elimination constraints*.

```
Definition sum_elim@{sl sr s s'|s ⇝ s'}
  {A : 𝒰@{sl}} {B : 𝒰@{sr}} {C : sum@{sl sr s} A B → 𝒰@{s'}}
  (u : sum@{sl sr s} A B)
```

The Missing Ingredient

Our proposal: interactions between sorts using *elimination constraints*.

```
Definition sum_elim@{sl sr s s'|s ⇝ s'}
  {A : 𝒰@{sl}} {B : 𝒰@{sr}} {C : sum@{sl sr s} A B → 𝒰@{s'}}
  (u : sum@{sl sr s} A B)  (f : forall (x : A), C (inl x))
  (g : forall (y : B), C (inr y)) : C u :=
  match u with
  | inl a ⇒ f a
  | inr b ⇒ g b
  end.
```

The Missing Ingredient

Our proposal: interactions between sorts using *elimination constraints*.

```
Definition sum_elim@{sl sr s s'|s ⇝ s'}
  {A : 𝒰@{sl}} {B : 𝒰@{sr}} {C : sum@{sl sr s} A B → 𝒰@{s'}}
  (u : sum@{sl sr s} A B)  (f : forall (x : A), C (inl x))
  (g : forall (y : B), C (inr y)) : C u :=
  match u with
  | inl a ⇒ f a
  | inr b ⇒ g b
  end.
```

Naturally models existing systems:

- Agda: `Set` $\rightsquigarrow$ `Set`, `Set` $\rightsquigarrow$ `Prop`, `Prop` $\rightsquigarrow$ `Prop`,
- Lean: `Type` $\rightsquigarrow$ `Type`, `Type` $\rightsquigarrow$ `Prop`, `Prop` $\rightsquigarrow$ `Prop`,
- Rocq: reflexive and transitive closure of `Type` $\rightsquigarrow$ `Prop`, `Prop` $\rightsquigarrow$ `SProp`.

I Plead Guilty

Elimination constraints are not so innocent:

```
Lemma guilty@{s | SProp ⇝ s} : forall {A : 𝒰@{s}} (x y : A), x = y.
Proof.
```

I Plead Guilty

Elimination constraints are not so innocent:

```
Lemma guilty@{s | SProp ⇝ s} : forall {A : 𝒰@{s}} (x y : A), x = y.
Proof.
  intros.
    (* A : 𝒰@{s}, x : A, y : A ⊢ x = y *)
```

I Plead Guilty

Elimination constraints are not so innocent:

```
Lemma guilty@{s | SProp ⇝ s} : forall {A : 𝒰@{s}} (x y : A), x = y.
Proof.
  intros.
    (* A : 𝒰@{s}, x : A, y : A ⊢ x = y *)
  set f : bool@{SProp} → A :=
    fun b ⇒ match b with true ⇒ x | false ⇒ y end.
    (* A : 𝒰@{s}, x : A, y : A, f : bool@{SProp} → A ⊢ x = y *)
```

I Plead Guilty

Elimination constraints are not so innocent:

```
Lemma guilty@{s | SProp ⇝ s} : forall {A : 𝒰@{s}} (x y : A), x = y.
Proof.
  intros.
    (* A : 𝒰@{s}, x : A, y : A ⊢ x = y *)
  set f : bool@{SProp} → A :=
    fun b ⇒ match b with true ⇒ x | false ⇒ y end.
    (* A : 𝒰@{s}, x : A, y : A, f : bool@{SProp} → A ⊢ x = y *)
  change (f true = f false).
    (* A : 𝒰@{s}, x : A, y : A, f : bool@{SProp} → A ⊢
          f true = f false *)
```

I Plead Guilty

Elimination constraints are not so innocent:

```
Lemma guilty@{s | SProp ⇝ s} : forall {A : 𝒰@{s}} (x y : A), x = y.
Proof.
  intros.
    (* A : 𝒰@{s}, x : A, y : A ⊢ x = y *)
  set f : bool@{SProp} → A :=
    fun b ⇒ match b with true ⇒ x | false ⇒ y end.
    (* A : 𝒰@{s}, x : A, y : A, f : bool@{SProp} → A ⊢ x = y *)
  change (f true = f false).
    (* A : 𝒰@{s}, x : A, y : A, f : bool@{SProp} → A ⊢
           f true = f false *)
  reflexivity.
    (* No more goals. *)
Qed.
```

I Plead Guilty

Elimination constraints are not so innocent:

```
Lemma guilty@{s | SProp ⇝ s} : forall {A : 𝒰@{s}} (x y : A), x = y.
Proof.
  intros.
    (* A : 𝒰@{s}, x : A, y : A ⊢ x = y *)
  set f : bool@{SProp} → A :=
    fun b ⇒ match b with true ⇒ x | false ⇒ y end.
    (* A : 𝒰@{s}, x : A, y : A, f : bool@{SProp} → A ⊢ x = y *)
  change (f true = f false).
    (* A : 𝒰@{s}, x : A, y : A, f : bool@{SProp} → A ⊢
          f true = f false *)
  reflexivity.
    (* No more goals. *)
Qed.
```

$\Longrightarrow$ for the typechecking to be right, have to inherit conversion rules!

A Funny Theorem

Theorem

Assuming SortPoly is consistent, then there does not exist a well-typed closed term $t : \bot^s$ in SortPoly$\tilde{\gamma}^{\rightarrow}$ for s a consistent sort.

A Funny Theorem

Theorem

Assuming SortPoly is consistent, then there does not exist a well-typed closed term $t : \bot^s$ in SortPoly$\widehat{\gamma}$ for s a consistent sort.

Precisions needed:

- Isn't this a tautology?
- What about $\mathbb{G}$?

A Funny Theorem

Theorem

Assuming SortPoly is consistent, then there does not exist a well-typed closed term $t : \bot^s$ in SortPoly$\widehat{\gamma}$ for s a consistent sort.

Precisions needed:

- Isn't this a tautology?
- What about $\mathbb{G}$?

Rapid-fire answers:

- No!
- Some conditions are needed.

A Funny Theorem

Theorem

Assuming SortPoly is consistent, then there does not exist a well-typed closed term $t : \bot^s$ in SortPoly$\widehat{\mathbb{Y}}$ for s a consistent sort.

Precisions needed:

- Isn't this a tautology?
- What about $\mathbb{G}$?

Rapid-fire answers:

- No!
- Some conditions are needed.

Let's talk about the required conditions.

Not So Easy

For a closed $t : \bot^s$, instantiate away sort variables.

Not So Easy

For a closed $t : \bot^s$, instantiate away sort variables. But how?

Not So Easy

For a closed $t : \bot^s$, instantiate away sort variables. But how?

- First attempt: an *initial*[2] sort `Data`.

[2] Initial in the sense that `Data` $\rightsquigarrow$ g for every ground sort g

Not So Easy

For a closed $t : \bot^s$, instantiate away sort variables. But how?

- First attempt: an *initial*[2] sort `Data`.
  Does not work: what if `SProp` $\rightsquigarrow$ `s'`?

[2] Initial in the sense that `Data` $\rightsquigarrow$ `g` for every ground sort `g`

Not So Easy

For a closed $t : \bot^s$, instantiate away sort variables. But how?

- First attempt: an *initial*[2] sort `Data`.
  Does not work: what if `SProp` $\rightsquigarrow$ `s'`?
- Second attempt: take minimal ground `g` s.t. `g` $\rightsquigarrow$ `s'`.

[2] Initial in the sense that `Data` $\rightsquigarrow$ `g` for every ground sort `g`

Not So Easy

For a closed $t : \bot^s$, instantiate away sort variables. But how?

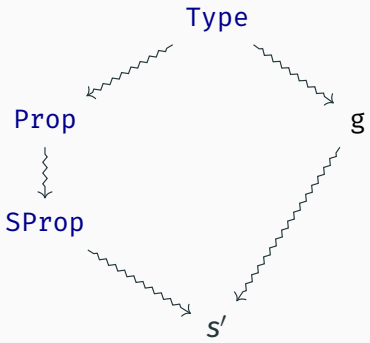
- First attempt: an *initial*[2] sort `Data`.
  Does not work: what if `SProp` $\rightsquigarrow$ `s'`?
- Second attempt: take minimal ground `g` s.t. `g` $\rightsquigarrow$ `s'`.
  Works, but not alone: need conditions.

[2] Initial in the sense that `Data` $\rightsquigarrow$ `g` for every ground sort `g`

Introducing Domination (ಠ_ಠ)

Second attempt: take minimal ground g s.t. $g \rightsquigarrow s'$.

Does not always exist!

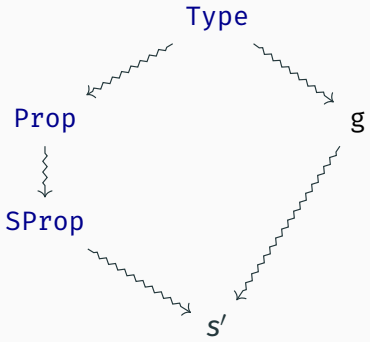


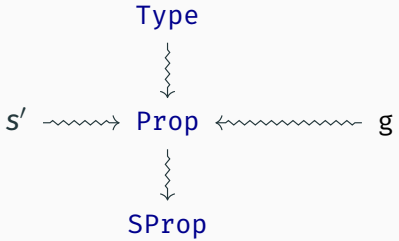
Introducing Domination (ಠ_ಠ)

Second attempt: take minimal ground g s.t. $g \rightsquigarrow s'$.

Does not always exist!

What about strays?



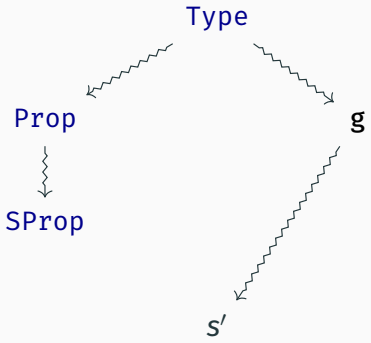


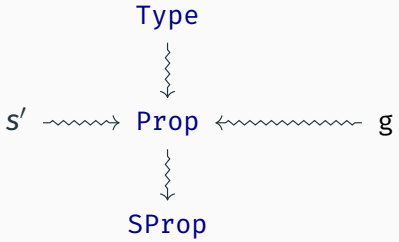
Introducing Domination (ಠ_ಠ)

Second attempt: take minimal ground g s.t. $g \rightsquigarrow s'$.

Does not always exist!
must be dominated!

What about strays?

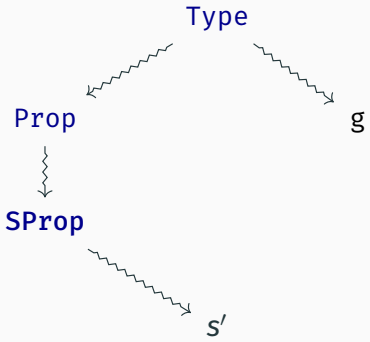




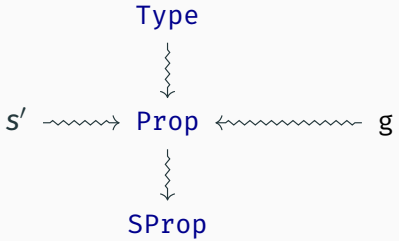
Introducing Domination (ಠ_ಠ)

Second attempt: take minimal ground g s.t. $g \rightsquigarrow s'$.

Does not always exist!
must be dominated!



What about strays?

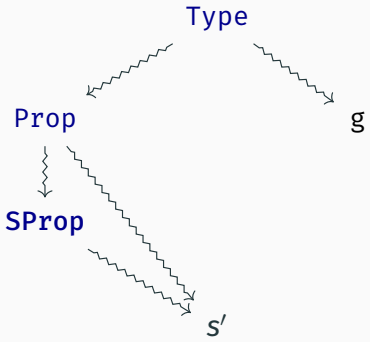


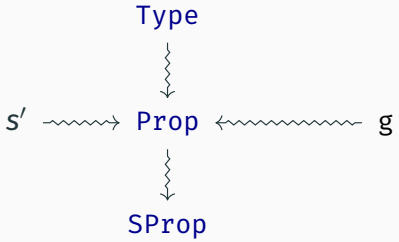
Introducing Domination (ಠ_ಠ)

Second attempt: take minimal ground g s.t. $g \rightsquigarrow s'$.

Does not always exist!
must be dominated!

What about strays?



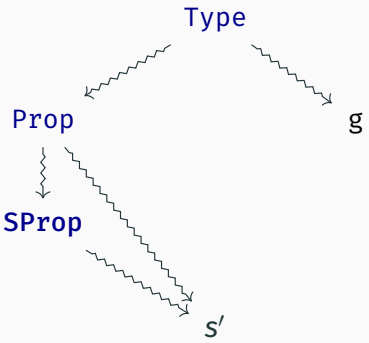


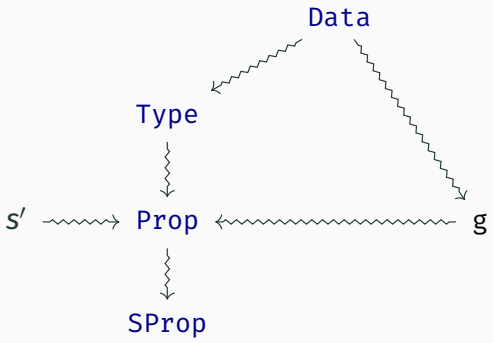
Introducing Domination (ಠ_ಠ)

Second attempt: take minimal ground g s.t. $g \rightsquigarrow s'$.

Does not always exist!
must be dominated!

What about strays?
have an initial sort `Data`





Let's End This Damn Proof

Theorem

If SortPoly is consistent, then there does not exist a well-typed closed term $t : \bot^s$ in SortPol$\tilde{\gamma}^{\rightarrow}$ for s a consistent sort.

For a closed $t : \bot^s$, instantiate away sort variables.

Let's End This Damn Proof

Theorem

If SortPoly is consistent, then there does not exist a well-typed closed term $t : \bot^s$ in SortPol$\tilde{\text{y}}^{\rightarrow}$ for s a consistent sort.

For a closed $t : \bot^s$, instantiate away sort variables. But how?

Let's End This Damn Proof

Theorem

If SortPoly is consistent, then there does not exist a well-typed closed term $t : \bot^s$ in SortPol$\tilde{\mathrm{y}}^{\rightarrow}$ for s a consistent sort.

For a closed $t : \bot^s$, instantiate away sort variables. But how?

- Ensure existence: all sort variables must be dominated.
- Take care of strays: initial sort.

Let's End This Damn Proof

Theorem

If SortPoly is consistent, then there does not exist a well-typed closed term $t : \bot^s$ in SortPol$\tilde{\tilde{y}}$ for s a consistent sort.

For a closed $t : \bot^s$, instantiate away sort variables. But how?

- Ensure existence: all sort variables must be dominated.
- Take care of strays: initial sort.

Under these conditions: instantiate by taking dominant ground sort (or initial sort if there are no dominants).

Rocq'n'Rolling

Implementation of elimination constraints in Rocq:

- ad-hoc checks for dominant sorts (amortized constant complexity),
- (current implementation restriction) manual prohibition of `SProp` $\rightsquigarrow$ `s`,
- elaboration to get principal sort assignment.

Some performance regressions ($\approx 4\%$) in the monomorphic case:

- eliminability check through a graph,
- bigger structures at elaboration.

Check out the RFC!

This is Goodbye (For Now)

We have extended the type system of SortPoly so that our extension:

- has relative consistency w.r.t SortPoly,
- has the same logical power as SortPoly,
- has a principal sort assignment,
- has an elaboration procedure yielding the principal sort.

This is Goodbye (For Now)

We have extended the type system of SortPoly so that our extension:

- has relative consistency w.r.t SortPoly,
- has the same logical power as SortPoly,
- has a principal sort assignment,
- has an elaboration procedure yielding the principal sort.

Future work:

- adapt tactics and libraries to bounded sort polymorphism,
- more metatheory, e.g., co-eliminability,
- add support in other proof assistants.

This is Goodbye (For Now)

We have extended the type system of SortPoly so that our extension:

- has relative consistency w.r.t SortPoly,
- has the same logical power as SortPoly,
- has a principal sort assignment,
- has an elaboration procedure yielding the principal sort.

Future work:

- adapt tactics and libraries to bounded sort polymorphism,
- more metatheory, e.g., co-eliminability,
- add support in other proof assistants.

Any question(s)?