

# Extending Sort Polymorphism with Elimination Constraints

TYPES2025, Glasgow

---

Tomás Díaz<sup>1</sup>   Johann Rosain<sup>2</sup>   Matthieu Sozeau<sup>2</sup>  
Nicolas Tabareau<sup>2</sup>   Théo Winterhalter<sup>3</sup>

June 10, 2025

<sup>1</sup>University of Chile, Chile

<sup>2</sup>LS2N & Inria de l'Université de Rennes, Nantes, France

<sup>3</sup>LMF & Inria Saclay, Saclay, France

## A Short Metastory

Real situation™

```
#[projections(primitive)] Record QVar : Type@{0} := mkQVar { val :  $\mathbb{N}$  }.
```

```
Definition Var (n :  $\mathbb{N}$ ) : QVar := { | val := n | }.
```

```
Definition eq_dec_qvar (x y : QVar) : { x = y } + { x  $\neq$  y } :=
```

```
  match eq_dec_ $\mathbb{N}$  (val x) (val y) with
```

```
  | inl e  $\Rightarrow$  inl (f_equal Var e)
```

```
  | inr f  $\Rightarrow$  inr (fun e  $\Rightarrow$  f (f_equal val e))
```

```
end.
```

# A Short Metastory

Real situation™

```
#[projections(primitive)] Record QVar : Type@{0} := mkQVar { val :  $\mathbb{N}$  }.
```

```
Definition Var (n :  $\mathbb{N}$ ) : QVar := { | val := n | }.
```

```
Definition eq_dec_qvar (x y : QVar) : { x = y } + { x  $\neq$  y } :=
```

```
  match eq_dec_ $\mathbb{N}$  (val x) (val y) with
```

```
  | inl e  $\Rightarrow$  inl (f_equal Var e)
```

```
  | inr f  $\Rightarrow$  inr (fun e  $\Rightarrow$  f (f_equal val e))
```

```
end.
```

# A Short Metastory

Real situation™

```
#[projections(primitive)] Record QVar : Type@{0} := mkQVar { val :  $\mathbb{N}$  }.
```

```
Definition Var (n :  $\mathbb{N}$ ) : QVar := { | val := n | }.
```

```
Definition eq_dec_qvar (x y : QVar) : { x = y } + { x  $\neq$  y } :=
```

```
  match eq_dec_ $\mathbb{N}$  (val x) (val y) with
```

```
  | inl e  $\Rightarrow$  inl (f_equal Var e)
```

```
  | inr f  $\Rightarrow$  inr (fun e  $\Rightarrow$  f (f_equal val e))
```

```
end.
```

# A Short Metastory

Real situation™

```
#[projections(primitive)] Record QVar : Type@{0} := mkQVar { val :  $\mathbb{N}$  }.
```

```
Definition Var (n :  $\mathbb{N}$ ) : QVar := { | val := n | }.
```

```
Definition eq_dec_qvar (x y : QVar) : { x = y } + { x  $\neq$  y } :=
```

```
  match eq_dec_ $\mathbb{N}$  (val x) (val y) with
```

```
  | inl e  $\Rightarrow$  inl (f_equal Var e)
```

```
  | inr f  $\Rightarrow$  inr (fun e  $\Rightarrow$  f (f_equal val e))
```

```
end.
```

**Error:** Found a constructor of inductive type sum while a constructor of sumbool is expected.

# A Short Metastory

Real situation™

```
#[projections(primitive)] Record QVar : Type@{0} := mkQVar { val :  $\mathbb{N}$  }.
```

```
Definition Var (n :  $\mathbb{N}$ ) : QVar := { | val := n | }.
```

```
Definition eq_dec_qvar (x y : QVar) : { x = y } + { x  $\neq$  y } :=
```

```
  match eq_dec_ $\mathbb{N}$  (val x) (val y) with
```

```
  | inleft e  $\Rightarrow$  inleft (f_equal Var e)
```

```
  | inright f  $\Rightarrow$  inright (fun e  $\Rightarrow$  f (f_equal val e))
```

```
end.
```

# A Short Metastory

Real situation™

```
#[projections(primitive)] Record QVar : Type@{0} := mkQVar { val :  $\mathbb{N}$  }.
```

```
Definition Var (n :  $\mathbb{N}$ ) : QVar := { | val := n | }.
```

```
Definition eq_dec_qvar (x y : QVar) : { x = y } + { x  $\neq$  y } :=
```

```
  match eq_dec_ $\mathbb{N}$  (val x) (val y) with
```

```
  | inleft e  $\Rightarrow$  inleft (f_equal Var e)
```

```
  | inright f  $\Rightarrow$  inright (fun e  $\Rightarrow$  f (f_equal val e))
```

```
end.
```

**Error:** Found a constructor of inductive type sumor while a constructor of sumbool is expected.

# A Short Metastory

Real situation™

```
#[projections(primitive)] Record QVar : Type@{0} := mkQVar { val :  $\mathbb{N}$  }.
```

```
Definition Var (n :  $\mathbb{N}$ ) : QVar := { | val := n | }.
```

```
Print sumbool.
```

```
Inductive sumbool (A B : Prop) : Type@{0} :=
```

```
| left : A → {A} + {B}
```

```
| right : B → {A} + {B}.
```



# A Short Metastory

Real situation™

```
#[projections(primitive)] Record QVar : Type@{0} := mkQVar { val :  $\mathbb{N}$  }.
```

```
Definition Var (n :  $\mathbb{N}$ ) : QVar := { | val := n | }.
```

```
Definition eq_dec_qvar (x y : QVar) : { x = y } + { x  $\neq$  y } :=
```

```
  match eq_dec_ $\mathbb{N}$  (val x) (val y) with
```

```
  | left e  $\Rightarrow$  left (f_equal Var e)
```

```
  | right f  $\Rightarrow$  right (fun e  $\Rightarrow$  f (f_equal val e))
```

```
end.
```

# A Short Metastory

Real situation™

```
#[projections(primitive)] Record QVar : Type@{0} := mkQVar { val :  $\mathbb{N}$  }.
```

```
Definition Var (n :  $\mathbb{N}$ ) : QVar := { | val := n | }.
```

```
Definition eq_dec_qvar (x y : QVar) : { x = y } + { x  $\neq$  y } :=
```

```
  match eq_dec_ $\mathbb{N}$  (val x) (val y) with
```

```
  | left e  $\Rightarrow$  left (f_equal Var e)
```

```
  | right f  $\Rightarrow$  right (fun e  $\Rightarrow$  f (f_equal val e))
```

```
end.
```

# A Short Metastory

Real situation™

```
#[projections(primitive)] Record QVar : Type@{0} := mkQVar { val :  $\mathbb{N}$  }.
```

```
Definition Var (n :  $\mathbb{N}$ ) : QVar := { | val := n | }.
```

```
Definition eq_dec_qvar (x y : QVar) : { x = y } + { x  $\neq$  y } :=  
  match eq_dec_ $\mathbb{N}$  (val x) (val y) with  
  | left e  $\Rightarrow$  left (f_equal Var e)  
  | right f  $\Rightarrow$  right (fun e  $\Rightarrow$  f (f_equal val e))  
end.
```

Pfiou.

# Why Would You Do That?

Rocq's sorts: **SProp**, **Prop**, **Type**.

Powerful, but at what cost?

```
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.
```

(This slide is powered by Rocq's Corelib©.)

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

```
Inductive sumor (A:Type) (B:Prop): Type :=  
| inleft: A → sumor A B  
| inright: B → sumor A B.
```

# Humanity's Savior: Jesus-Christ Sort Polymorphism

Since Coq 8.19:

```
Inductive psum@{sl sr s ; ul ur} (A :  $\mathcal{U}$ @{sl ; ul}) (B :  $\mathcal{U}$ @{sr ; ur}) :  $\mathcal{U}$ @{s ; max(ul,ur)} :=  
| inl : A  $\rightarrow$  psum A B  
| inr : B  $\rightarrow$  psum A B.
```

Recall: a sort is a quotiented pair (quality, universe)

```
Definition sum@{u v} := psum@{Type Type Type ; u v}.  
(* sum : Type@{u}  $\rightarrow$  Type@{v}  $\rightarrow$  Type@{max(u,v)} *)  
Definition or := psum@{Prop Prop Prop ; 0 0}.  
(* or : Prop  $\rightarrow$  Prop  $\rightarrow$  Prop *)  
Definition sumbool := psum@{Prop Prop Type ; 0 0}.  
(* sumbool : Prop  $\rightarrow$  Prop  $\rightarrow$  Type@{0} *)  
Definition sumor@{u} := psum@{Type Prop Type ; u 0}.  
(* sumor : Type@{u}  $\rightarrow$  Prop  $\rightarrow$  Type@{u} *)
```

# Humanity's Savior: Jesus-Christ Sort Polymorphism

Since Coq 8.19:

```
Inductive psum@{sl sr s ; ul ur} (A :  $\mathcal{U}$ @{sl ; ul}) (B :  $\mathcal{U}$ @{sr ; ur}) :  $\mathcal{U}$ @{s ; max(ul,ur)} :=  
| inl : A  $\rightarrow$  psum A B  
| inr : B  $\rightarrow$  psum A B.
```

Recall: a sort is a quotiented pair (quality, universe)

```
Definition sum@{u v} := psum@{Type Type Type ; u v}.  
(* sum : Type@{u}  $\rightarrow$  Type@{v}  $\rightarrow$  Type@{max(u,v)} *)
```

```
Definition or := psum@{Prop Prop Prop ; 0 0}.
```

```
(* or : Prop  $\rightarrow$  Prop  $\rightarrow$  Prop *)
```

```
Definition sumbool := psum@{Prop Prop Type ; 0 0}.
```

```
(* sumbool : Prop  $\rightarrow$  Prop  $\rightarrow$  Type@{0} *)
```

```
Definition sumor@{u} := psum@{Type Prop Type ; u 0}.
```

```
(* sumor : Type@{u}  $\rightarrow$  Prop  $\rightarrow$  Type@{u} *)
```

psum is defined

Wait, what?

## Mild Disappointment

```
Definition psum_elim@{sl sr s s' ; ul ur}  
  {A :  $\mathcal{U}$ @{sl ; ul}} {B :  $\mathcal{U}$ @{sr ; ur}} {C :  $\mathcal{U}$ @{s' ; max(ul, ur)}}  
  (f : A  $\rightarrow$  C) (g : B  $\rightarrow$  C) (x : psum@{sl sr s ; ul ur} A B) :=  
  match x with  
  | inl a  $\Rightarrow$  f a  
  | inr b  $\Rightarrow$  g b  
end.
```

## Mild Disappointment

```
Definition psum_elim@{sl sr s s' ; ul ur}  
  {A :  $\mathcal{U}$ @{sl ; ul}} {B :  $\mathcal{U}$ @{sr ; ur}} {C :  $\mathcal{U}$ @{s' ; max(ul, ur)}}  
  (f : A  $\rightarrow$  C) (g : B  $\rightarrow$  C) (x : psum@{sl sr s ; ul ur} A B) :=  
  match x with  
  | inl a  $\Rightarrow$  f a  
  | inr b  $\Rightarrow$  g b  
end.
```

**Error:** Elimination of a sort polymorphic inductive object instantiated to a variable sort quality is only allowed on a predicate in the same sort quality.



## Mild Disappointment

```
Definition psum_elim@{sl sr s s' ; ul ur}  
  {A :  $\mathcal{U}$ @{sl ; ul}} {B :  $\mathcal{U}$ @{sr ; ur}} {C :  $\mathcal{U}$ @{s' ; max(ul, ur)}}  
  (f : A  $\rightarrow$  C) (g : B  $\rightarrow$  C) (x : psum@{sl sr s ; ul ur} A B) :=  
  match x with  
  | inl a  $\Rightarrow$  f a  
  | inr b  $\Rightarrow$  g b  
end.
```

**Error:** Elimination of a sort polymorphic inductive object instantiated to a variable sort quality is only allowed on a predicate in the same sort quality.

$\Rightarrow$  Have to declare eliminators by hand...<sup>1</sup>

---

<sup>1</sup>With only **Prop** and **Type**, we could do something using bad dark magic though.

# Today's Talk Objective

```
Definition psum_elim@{sl sr s s' ; ul ur} {A :  $\mathcal{U}$ @{sl ; ul}}  
  {B :  $\mathcal{U}$ @{sr ; ur}} {C :  $\mathcal{U}$ @{s' ; max(ul,ur)}} (f : A → C) (g : B → C)  
  (x : psum@{sl sr s ; ul ur} A B) :=  
  match x with  
  | inl a ⇒ f a  
  | inr b ⇒ g b  
  end.
```

```
Definition sumbool_rect := psum_elim@{Prop Prop Type Type; 0 0}.
```

```
Definition sumbool_ind := psum_elim@{Prop Prop Type Prop; 0 0}.
```

```
Definition sumbool_sind := psum_elim@{Prop Prop Type SProp; 0 0}.
```

```
Fail Definition or_rect := psum_elim@{Prop Prop Prop Type; 0 0}.
```

The command has indeed failed with message: cannot create computational content from proof.

# Overview of Elimination in Rocq's Type Theory

Binary relation “eliminates to”:

- ▶ **Type** eliminates to **Prop**,
- ▶ **Type** eliminates to **SProp**,
- ▶ **Prop** eliminates to **SProp**,
- ▶ **s** eliminates to **s**.

$$\frac{\dots \quad x:A \quad A:s \quad P:s' \quad s \text{ eliminates to } s'}{\text{match } x \text{ return } P \text{ with } \dots \text{ end well typed}}$$
$$\frac{\dots \quad x:A \quad A:s \quad P:s' \quad s \text{ eliminates to } s' \text{ or } s = \text{Prop}}{\text{fix } F \text{ x } \vec{y} \{ \text{struct } x \} : P := \dots \text{ well typed}}$$

# Rocq's (Updated) Type Theory

Populate partial order  $\rightsquigarrow$ :

User-defined  $s \rightsquigarrow s'$ ,       $s$  eliminates to  $s' \implies s \rightsquigarrow s'$ .

# Rocq's (Updated) Type Theory

Populate partial order  $\rightsquigarrow$ :

User-defined  $s \rightsquigarrow s'$ ,  $s$  eliminates to  $s' \implies s \rightsquigarrow s'$ .

$$\frac{\dots \quad x:A \quad A:s \quad P:s' \quad s \rightsquigarrow s'}{\text{match } x \text{ return } P \text{ with } \dots \text{ end well typed}}$$
$$\frac{\dots \quad x:A \quad A:s \quad P:s' \quad s \rightsquigarrow s' \text{ or } s \rightsquigarrow \text{Prop}}{\text{fix } F \text{ x } \vec{y} \{ \text{struct } x \}: P := \dots \text{ well typed}}$$

# For Once, It's Not My Fault

Definition  $\text{foo}\{s \mid s \rightsquigarrow \text{Prop}\} \dots := \dots$

Definition  $\text{bar}\{s' \ s'' \mid s' \rightsquigarrow s''\} \dots := \dots$

Sort  $\text{Exc.} \ (* \text{ Global sort } *)$

Axiom  $\text{raise} : \forall A : \text{Exc}, A.$

Constraint  $\text{Type} \rightsquigarrow \text{Exc.}$

Definition  $\text{foo}'\{s \ s' \mid \text{Exc} \rightsquigarrow s, s \rightsquigarrow s', s' \rightsquigarrow \text{Prop}\} :=$   
...  $\text{foo}\{s'\}$  ...  $\text{bar}\{s \ s'\}$  ...

Type



Prop



SProp

# For Once, It's Not My Fault

Definition  $\text{foo@}\{s \mid s \rightsquigarrow \text{Prop}\} \dots := \dots$

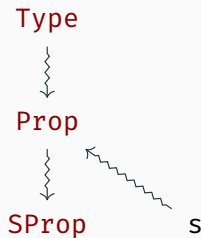
Definition  $\text{bar@}\{s' \ s'' \mid s' \rightsquigarrow s''\} \dots := \dots$

Sort  $\text{Exc.} \ (* \ \text{Global sort} \ *)$

Axiom  $\text{raise} : \forall A : \text{Exc}, A.$

Constraint  $\text{Type} \rightsquigarrow \text{Exc.}$

Definition  $\text{foo'@}\{s \ s' \mid \text{Exc} \rightsquigarrow s, s \rightsquigarrow s', s' \rightsquigarrow \text{Prop}\} :=$   
 $\dots \text{foo@}\{s'\} \dots \text{bar@}\{s \ s'\} \dots$



# For Once, It's Not My Fault

Definition  $\text{foo@}\{s \mid s \rightsquigarrow \text{Prop}\} \dots := \dots$

Definition  $\text{bar@}\{s' \ s'' \mid s' \rightsquigarrow s''\} \dots := \dots$

Sort  $\text{Exc.} \ (* \text{ Global sort } *)$

Axiom  $\text{raise} : \forall A : \text{Exc}, A.$

Constraint  $\text{Type} \rightsquigarrow \text{Exc.}$

Definition  $\text{foo'@}\{s \ s' \mid \text{Exc} \rightsquigarrow s, s \rightsquigarrow s', s' \rightsquigarrow \text{Prop}\} :=$   
...  $\text{foo@}\{s'\}$  ...  $\text{bar@}\{s \ s'\}$  ...

Type



Prop



SProp



s

s'



s''



# For Once, It's Not My Fault

Definition  $\text{foo}\@{s|s \rightsquigarrow \text{Prop}} \dots := \dots$

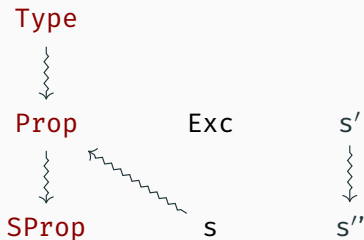
Definition  $\text{bar}\@{s' s''|s' \rightsquigarrow s''} \dots := \dots$

Sort  $\text{Exc.}$  (\* Global sort \*)

Axiom  $\text{raise} : \forall A : \text{Exc}, A.$

Constraint  $\text{Type} \rightsquigarrow \text{Exc}.$

Definition  $\text{foo}'\@{s s'|\text{Exc} \rightsquigarrow s, s \rightsquigarrow s', s' \rightsquigarrow \text{Prop}} :=$   
...  $\text{foo}\@{s'} \dots \text{bar}\@{s s'} \dots$



# For Once, It's Not My Fault

Definition  $\text{foo@}\{s \mid s \rightsquigarrow \text{Prop}\} \dots := \dots$

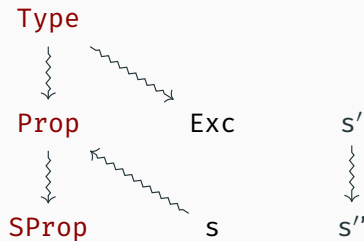
Definition  $\text{bar@}\{s' \ s'' \mid s' \rightsquigarrow s''\} \dots := \dots$

Sort  $\text{Exc.} \text{ (* Global sort *)}$

Axiom  $\text{raise} : \forall A : \text{Exc}, A.$

Constraint  $\text{Type} \rightsquigarrow \text{Exc.}$

Definition  $\text{foo'@}\{s \ s' \mid \text{Exc} \rightsquigarrow s, s \rightsquigarrow s', s' \rightsquigarrow \text{Prop}\} :=$   
 $\dots \text{foo@}\{s'\} \dots \text{bar@}\{s \ s'\} \dots$



# For Once, It's Not My Fault

Definition  $\text{foo@}\{s \mid s \rightsquigarrow \text{Prop}\} \dots := \dots$

Definition  $\text{bar@}\{s' \ s'' \mid s' \rightsquigarrow s''\} \dots := \dots$

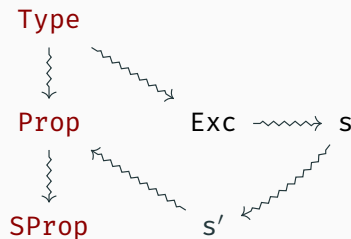
Sort  $\text{Exc.} (* \text{ Global sort } *)$

Axiom  $\text{raise} : \forall A : \text{Exc}, A.$

Constraint  $\text{Type} \rightsquigarrow \text{Exc}.$

Definition  $\text{foo'@}\{s \ s' \mid \text{Exc} \rightsquigarrow s, s \rightsquigarrow s', s' \rightsquigarrow \text{Prop}\} :=$   
 $\dots \text{foo@}\{s'\} \dots \text{bar@}\{s \ s'\} \dots$

**Error:** Elimination constraints imply an undeclared elimination between  $\text{Exc}$  and  $\text{Prop}$ .



# For Once, It's Not My Fault

```
Definition foo@{s | s  $\rightsquigarrow$  Prop} ... := ...
```

```
Definition bar@{s' s'' | s'  $\rightsquigarrow$  s''} ... := ...
```

```
Sort Exc. (* Global sort *)
```

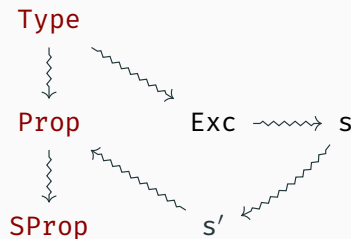
```
Axiom raise :  $\forall$  A : Exc, A.
```

```
Constraint Type  $\rightsquigarrow$  Exc.
```

```
Definition foo'@{s s' | Exc  $\rightsquigarrow$  s, s  $\rightsquigarrow$  s', s'  $\rightsquigarrow$  Prop} :=  
... foo@{s'} ... bar@{s s'} ...
```

**Error:** Elimination constraints imply an undeclared elimination between Exc and Prop.

Elimination in (S)Prop: consistency...



# (Im)Predicativity

## Definition

A sort  $s$  is impredicative if there is a term of type  $\mathcal{U}@{s;u} \simeq \mathcal{U}@{s;v}$ .

$SProp$  &  $Prop$  are impredicative.

Elimination	Provable properties
$Prop \rightsquigarrow s$	$s$ impredicative
$SProp \rightsquigarrow s$	$s$ impredicative & prop. proof irrelevant

But  $s$  predicative: could be  $Type$ ...

# (Im)Predicativity

## Definition

A sort  $s$  is impredicative if there is a term of type  $\mathcal{U}@{s;u} \simeq \mathcal{U}@{s;v}$ .

$SProp$  &  $Prop$  are impredicative.

Elimination	Provable properties
$Prop \rightsquigarrow s$	$s$ impredicative
$SProp \rightsquigarrow s$	$s$ impredicative & prop. proof irrelevant

But  $s$  predicative: could be  $Type$ ...

~~$Prop \rightsquigarrow s$~~

~~$SProp \rightsquigarrow s$~~

# What If It Was Allowed?

Monomorphization theorem: for every

**Definition**  $\text{foo} @ \{s \mid s \rightsquigarrow s'\} := \dots$

duplicate for *ground*  $s, s'$  s.t.  $s \rightsquigarrow s'$ .

- ▶ Everything is well typed.
- ▶ Implies *ground* equiconsistency.

# What If It Was Allowed?

Monomorphization theorem: for every

**Definition**  $\text{foo} \hat{=} \{s \mid s \rightsquigarrow s'\} := \dots$

duplicate for *ground*  $s, s'$  s.t.  $s \rightsquigarrow s'$ .

- ▶ Everything is well typed.
- ▶ Implies *ground* equiconsistency.

With global sorts:

- ▶ **SProp**  $\rightsquigarrow s$ : acceptable.
- ▶ **Prop**  $\rightsquigarrow s$ : under investigation.
- ▶ User-defined inconsistencies (e.g., **Exc** and **Exc**  $\rightsquigarrow$  **Prop**).



## Status of the Current Implementation

Can write  $s \rightsquigarrow s'$  for *local* and *global* sorts. But:

	Current status	Solution?
Rigid sorts	User specifies all constraints	Doing otherwise is unsound Consistency is user's responsibility
$\text{Prop} \rightsquigarrow s$	Prohibited	Predicativity mark (sufficient?)
$\text{SProp} \rightsquigarrow s$	Prohibited	Definitional PI + predicativity mark

## Status of the Current Implementation

Can write  $s \rightsquigarrow s'$  for *local* and *global* sorts. But:

	Current status	Solution?
Rigid sorts	User specifies all constraints	Doing otherwise is unsound Consistency is user's responsibility
$\text{Prop} \rightsquigarrow s$	Prohibited	Predicativity mark (sufficient?)
$\text{SProp} \rightsquigarrow s$	Prohibited	Definitional PI + predicativity mark

### Remark

(Sub)singleton elimination is not handled in the framework.

## The Case of Cases

(Sub)singleton elimination: inductives with  $\leq 1$  constructor:  $\perp$ , Accessibility, equality, conjunction, ...

Long story short: I've bluffed you.

$$\frac{\begin{array}{c} \dots \quad x:A \quad A:s \quad P:s' \\ s \rightsquigarrow s' \end{array}}{\text{match } x \text{ return } P \text{ with } \dots \text{ end well typed}}$$

## The Case of Cases

(Sub)singleton elimination: inductives with  $\leq 1$  constructor:  $\perp$ , Accessibility, equality, conjunction, ...

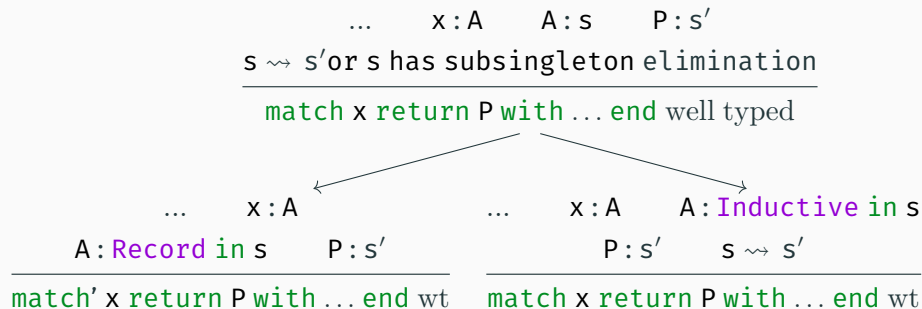
Long story short: I've bluffed you.

$$\frac{\begin{array}{c} \dots \quad x:A \quad A:s \quad P:s' \\ s \rightsquigarrow s' \text{ or } s \text{ has subsingleton elimination} \end{array}}{\text{match } x \text{ return } P \text{ with } \dots \text{ end well typed}}$$

## The Case of Cases

(Sub)singleton elimination: inductives with  $\leq 1$  constructor:  $\perp$ , **Accessibility**, equality, conjunction, ...

Long story short: I've bluffed you.



# Conclusion

We have:

- ▶ Implemented elimination constraints in Rocq.
- ▶ Enabled writing the most generic eliminator in Rocq.
- ▶ Proved consistency of system (on paper, for *ground* sorts).

```
Definition psum_elim @ {sl sr s s' ; ul ur | s ~> s'} {A :  $\mathcal{U}$ @{sl ; ul}}
```

```
  {B :  $\mathcal{U}$ @{sr ; ur}} {C :  $\mathcal{U}$ @{s' ; max(ul, ur)}} (f : A → C) (g : B → C)
```

```
  (x : psum @ {sl sr s ; ul ur} A B) :=
```

```
  match x with
```

```
  | inl a ⇒ f a
```

```
  | inr b ⇒ g b
```

```
end.
```

# Conclusion

We have:

- ▶ Implemented elimination constraints in Rocq.
- ▶ Enabled writing the most generic eliminator in Rocq.
- ▶ Proved consistency of system (on paper, for *ground* sorts).

```
Definition psum_elim@{sl sr s s' ; ul ur | s  $\rightsquigarrow$  s'} {A :  $\mathcal{U}$ @{sl ; ul}}  
  {B :  $\mathcal{U}$ @{sr ; ur}} {C :  $\mathcal{U}$ @{s' ; max(ul, ur)}} (f : A  $\rightarrow$  C) (g : B  $\rightarrow$  C)  
  (x : psum@{sl sr s ; ul ur} A B) :=  
  match x with  
  | inl a  $\Rightarrow$  f a  
  | inr b  $\Rightarrow$  g b  
end.
```



# Conclusion

We have:

- ▶ Implemented elimination constraints in Rocq.
- ▶ Enabled writing the most generic eliminator in Rocq.
- ▶ Proved consistency of system (on paper, for *ground* sorts).

We plan to:

- ▶ Add elimination constraints in MetaRocq (current status: formalizing sort polymorphism).
- ▶ Allow  $SProp \rightsquigarrow s$ .
- ▶ Allow  $Prop \rightsquigarrow s$  if consistent.



# Conclusion

We have:

- ▶ Implemented elimination constraints in Rocq.
- ▶ Enabled writing the most generic eliminator in Rocq.
- ▶ Proved consistency of system (on paper, for *ground* sorts).

We plan to:

- ▶ Add elimination constraints in MetaRocq (current status: formalizing sort polymorphism).
- ▶ Allow  $\text{SProp} \rightsquigarrow s$ .
- ▶ Allow  $\text{Prop} \rightsquigarrow s$  if consistent.

Thank you for your attention!