Theory and Metatheory of Elimination in Rocq

Master 2 Internship Defense, Lyon

Johann Rosain[1]
Supervised by: Matthieu Sozeau[2] and Théo Winterhalter [3]

July 10, 2025

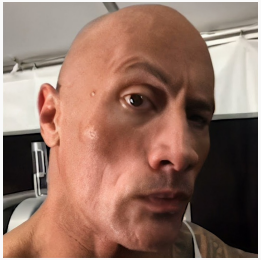
[1]École Normale Supérieure de Lyon, Lyon, France

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

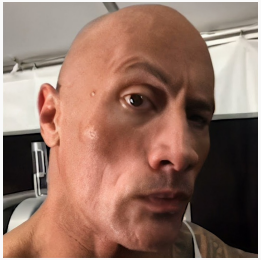
[3]LMF & Inria Saclay, Saclay, France

Là il y a Pierre, Pierre, Pierre et Pierre. Mais on l'appelle le roux, lui

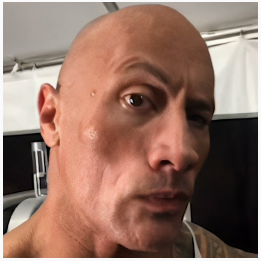
Là il y a Pierre, Pierre, Pierre et Pierre. Mais on l'appelle le roux, lui



Là il y a Pierre, Pierre, Pierre et Pierre. Mais on l'appelle le roux, lui



Là il y a Pierre, Pierre, Pierre et Pierre. Mais on l'appelle le roux, lui



Liberté, égalité, prouvabilité

Avoid duplication in proof assistants.

Avoid duplication in proof assistants.

Proof assistants?

- Model your problem.
- Try to prove things about it.
- Get a cold, mechanical, answer.[1]

[1] Or: `Anomaly: *some non-understandable gibberish*, please report.`

Avoid duplication in proof assistants.

Proof assistants?

- Model your problem.
- Try to prove things about it.
- Get a cold, mechanical, answer.[1]

Why is this a big deal?

- Modularity: easily add features.
- Increases robustness.

[1] Or: `Anomaly: *some non-understandable gibberish*, please report.`

Liberté, égalité, prouvabilité

Avoid duplication in proof assistants.

Proof assistants?

- Model your problem.
- Try to prove things about it.
- Get a cold, mechanical, answer.[1]

Why is this a big deal?

- Modularity: easily add features.
- Increases robustness.

Rocq is a proof assistant doing things ~~mostly~~ right.

[1] Or: `Anomaly: *some non-understandable gibberish*, please report.`

Le duché de Tiffe

Today's talk will mostly focus on

Concept used in every-day life:

- Program instructions.
- Paths in a graph (or free categories).

Today's talk will mostly focus on

inductives.

Concept used in every-day life:

- Program instructions.
- Paths in a graph (or free categories).

Or to write a sum type (i.e., a coproduct).

$$\frac{\Gamma \vdash x : A}{\Gamma \vdash \mathtt{inl}\ x : A + B} \qquad \frac{\Gamma \vdash y : B}{\Gamma \vdash \mathtt{inr}\ y : A + B}$$

Today's talk will mostly focus on

inductives.

Concept used in every-day life:

- Program instructions.
- Paths in a graph (or free categories).

Or to write a sum type (i.e., a coproduct).

$$\frac{\Gamma \vdash x : A}{\Gamma \vdash \mathtt{inl}\ x : A + B} \qquad \frac{\Gamma \vdash y : B}{\Gamma \vdash \mathtt{inr}\ y : A + B}$$

What could *possibly* go wrong?

Comme une sensation de déjà vu...

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

Comme une sensation de déjà vu...

```
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.
```

Comme une sensation de déjà vu...

```
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.

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

Comme une sensation de déjà vu...

```
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.

Inductive sumbool (A B : Prop) : Type :=
| 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.
```

Comme une sensation de déjà vu...

```
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.

Inductive sumbool (A B : Prop) : Type :=
| 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.
```

“Prouver plus, c’est mieux”

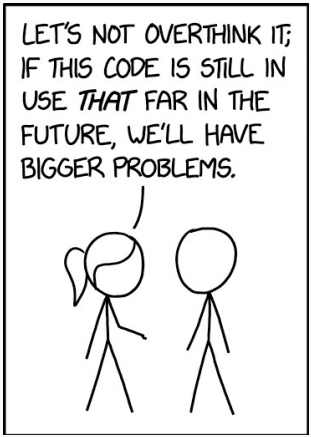
In Rocq, terms live in sorts (`Prop`, `Type`).

Fine when there are 2 sorts: “only” 8 possible combinations.

“Prouver plus, c’est mieux”

In Rocq, terms live in sorts (`Prop`, `Type`).

Fine when there are 2 sorts: “only” 8 possible combinations.



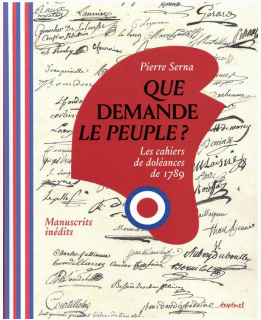
Rare image of the Rocq’s development team, 2005 (xkcd.com/2730)

On en a gros

Spoiler: we **are** *that* far in the future!

On en a gros

Spoiler: we **are** *that* far in the future!



Users ask for more!

- Strict propositions,
- Fibrant or strict universes (HoTT),
- Pure or impure universes (RETT),
- Proof relevance with computational irrelevance (GTT),
- etc...

Gi rarement vu ça

Recall our initial goal:

Gi rarement vu ça

Recall our initial goal:

Avoid duplication in proof assistants.

How do we go about that?

Gi rarement vu ça

Recall our initial goal:

Avoid duplication in proof assistants.

How do we go about that?

We are type theoreticians, so the answer is clear!

Gi rarement vu ça

Recall our initial goal:

Avoid duplication in proof assistants.

How do we go about that?

We are type theoreticians, so the answer is clear!

Il vous faudra autre chose?

Courtesy of J. Poiret et al (2025), you can write:

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

Il vous faudra autre chose?

Courtesy of J. Poiret et al (2025), you can write:

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

Cool! There is no more duplication then!

Il vous faudra autre chose?

Courtesy of J. Poiret et al (2025), you can write:

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

~~Cool! There is no more duplication then!~~

Il vous faudra autre chose?

Courtesy of J. Poiret et al (2025), you can write:

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

Cool! There is no more duplication then!

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

Il vous faudra autre chose?

Courtesy of J. Poiret et al (2025), you can write:

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

Cool! There is no more duplication then!

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

Il vous faudra autre chose?

Courtesy of J. Poiret et al (2025), you can write:

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

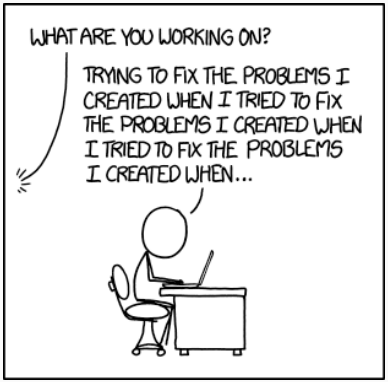
~~Cool! There is no more duplication then!~~

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

Still need to declare 2^n induction principles...

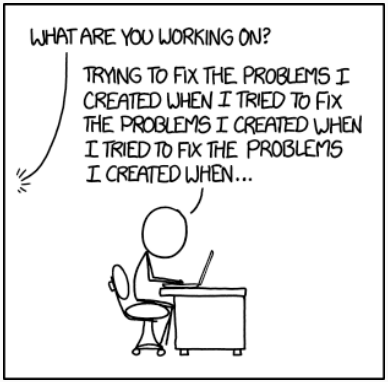
Why is it failing?

Why is it failing?



Cannot create a `Type`-valued sum from a logical disjunction.

Why is it failing?

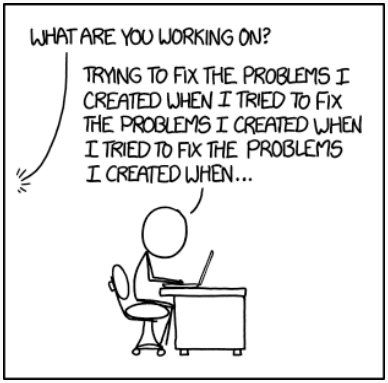


Cannot create a `Type`-valued sum from a logical disjunction.

Otherwise, excluded middle makes everything decidable.[2]

[2] You don’t need me to tell you that this would be bad!!

Why is it failing?



Cannot create a `Type`-valued sum from a logical disjunction.

Otherwise, excluded middle makes everything decidable.[2]

With ~~sort polymorphism~~, no way to express that.

[2] You don’t need me to tell you that this would be bad!!

Elle est fraiche, elle est fraiche ma théorie !

So, how do we actually

Avoid duplication in proof assistants?

So, how do we actually

Avoid duplication in proof assistants?

Meet our new, shiny, theory:

$\mathrm{SortPoly}^{\rightarrow}$

So, how do we actually

Avoid duplication in proof assistants?

Meet our new, shiny, theory:

$\mathrm{SortPoly}^{\rightarrow}$

Main idea:

Introduce elimination constraints.

Il fallait la trouver, cette idée

Main idea:

Introduce elimination constraints $s \rightsquigarrow s'$.

Il fallait la trouver, cette idée

Main idea:

Introduce elimination constraints $s \rightsquigarrow s'$.

```
Sort sl sr s s'.
Constraint s ⇝ s'.

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

Il fallait la trouver, cette idée

Main idea:

Introduce elimination constraints $s \rightsquigarrow s'$.

```
Sort sl sr s s'.
Constraint s ⇝ s'.

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

Il fallait la trouver, cette idée

Main idea:

Introduce elimination constraints $s \rightsquigarrow s'$.

```
Sort sl sr s s'.
Constraint s ⇝ s'.

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

Il fallait la trouver, cette idée

Main idea:

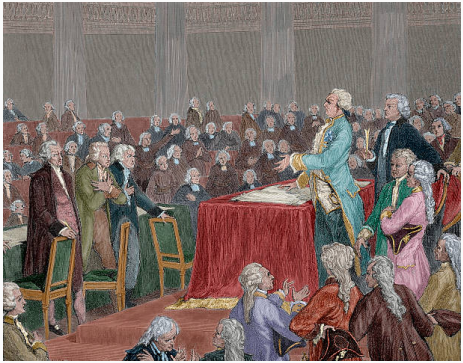
Introduce elimination constraints $s \rightsquigarrow s'$.

```
Sort sl sr s s'.
Constraint s ⇝ s'.

Definition sum_elim {A : Uα{sl}} {B : Uα{sr}} {C : Uα{s'}}
 (f : A → C) (g : B → C) (u : sumα{sl sr s} A B) : C :=
 match u with
 | inl a ⇒ f a
 | inr b ⇒ g b
 end.
```

Cool, but… is that it?

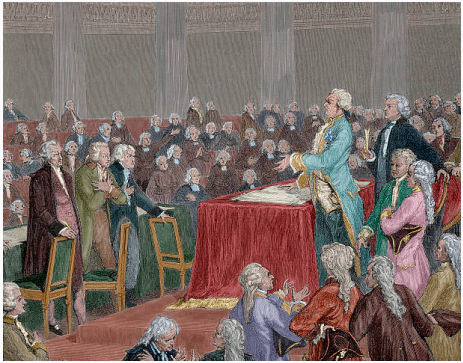
Les doléances du peuple



The General Users

Want it in Rocq.

Les doléances du peuple



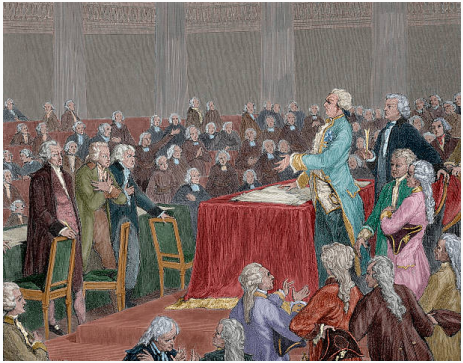
The General Users

Want it in Rocq.

The Logicians

There is no proof $\vdash t : \bot$.

Les doléances du peuple



The General Users

Want it in Rocq.

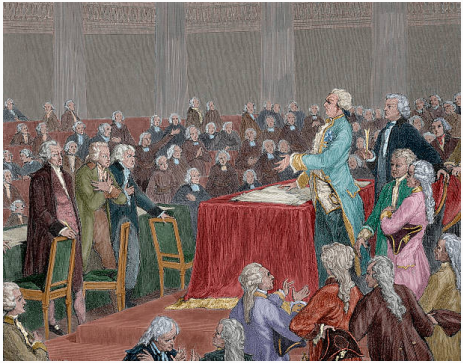
The Logicians

There is no proof $\vdash t : \bot$.

Rocq's Maintainers

Type-checking is decidable

Les doléances du peuple



The General Users

Want it in Rocq.

The Logicians

There is no proof $\vdash t : \bot$.

Rocq's Maintainers

Type-checking is decidable

This is pretty hard.

L’arnaqueur arnaqué

In the end, what did we get?

- Implemented in Rocq, in the process of being merged.

L'arnaqueur arnaqué

In the end, what did we get?

- Implemented in Rocq, in the process of being merged.
- Consistency of $\mathrm{SortPol}_{\vec{\mathbb{Y}}}$ under closure condition.

L'arnaqueur arnaqué

In the end, what did we get?

- Implemented in Rocq, in the process of being merged.
- Consistency of $\mathrm{SortPol}_{\tilde{\mathrm{y}}}^{\rightarrow}$ under closure condition.
- Type-checking probably decidable under stronger conditions.

L'arnaqueur arnaqué

In the end, what did we get?

- Implemented in Rocq, in the process of being merged.
- Consistency of $\mathrm{SortPoly}^{\vec{\forall}}$ under closure condition.
- Type-checking probably decidable under stronger conditions.

No general theorem, but works for Rocq's Type Theory.

“Au revoir.”

Avoid duplication in proof assistants by using $\mathrm{SortPoly}^{\overrightarrow{\sim}}$.

“Au revoir.”

Avoid duplication in proof assistants by using $\mathrm{SortPoly}^{\overrightarrow{\forall}}$.

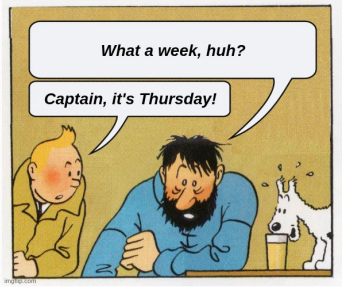
Recap:

- Implemented in Rocq (8k lines).
- Pen-and-paper proof of consistency.
- Start of formalization in MetaRocq (6k lines).
- System presented at TYPES 2025.
- More comprehensive writ: POPL 2026.

Avoid duplication in proof assistants by using $\mathrm{SortPoly}^{\rightrightarrows}$.

Recap:

- Implemented in Rocq (8k lines).
- Pen-and-paper proof of consistency.
- Start of formalization in MetaRocq (6k lines).
- System presented at TYPES 2025.
- More comprehensive writ: POPL 2026.



Thanks for your attention, any question?