# A Ghost Sort for Proof-Relevant yet Erased Data in Rocq and MetaRocq

TYPES2025, Glasgow

Johann Rosain[1] Matthieu Sozeau[1] Théo Winterhalter [2]

June 13, 2025

[1] LS2N & Inria de l'Université de Rennes, Nantes, France

[2] LMF & Inria Saclay, Saclay, France

# It Sounds Like a Bad Advertisement

Are you:

- Often using Rocq’s extraction?
- Annoyed by the slowdown of unnecessary arguments?

# It Sounds Like a Bad Advertisement

Are you:

- Often using Rocq's extraction?
- Annoyed by the slowdown of unnecessary arguments?

If so, then ~~use~~buy our new ghost sort!

- Takes care of 99.9% of the inconveniences of extraction!
- At the cheap price of a single annotation!

Otherwise, you may still find the talk interesting, I swear (spoiler: there will be drama)!

# Ghost Sort In Action

```
Fixpoint lookup {A} (l : list A) (f : fin (length l)) : A := match l with
 | nil ⇒ False_rect A f
 | cons a l ⇒ match f with
   | finO ⇒ a
   | finS f’ ⇒ lookup l f’
   end
 end.
```

```
let rec lookup l f =
 match l with
 | Nil → assert false
 | Cons (a, l0) → (match f with
                   | FinO _ → a
                   | FinS (_, f’) → lookup l0 f’)
```

# Ghost Sort In Action

```
Fixpoint lookup {A} (l : list A) (f : fin (length l)) : A := match l with
 | nil ⇒ False_rect A f
 | cons a l ⇒ match f with
   | finO ⇒ a
   | finS f' ⇒ lookup l f'
   end
 end.
```

```
let rec lookup l f =
 match l with
 | Nil → assert false
 | Cons (a, l0) → (match f with
                 | FinO _ → a
                 | FinS (_, f') → lookup l0 f')
```

useless!

# Ghost Sort In Action

```
Fixpoint lookup {A} (l : list A) (f : fin (length@{Ghost} l)) : A := match l with
 | nil ⇒ False_rect A f
 | cons a l ⇒ match f with
   | finO ⇒ a
   | finS f' ⇒ lookup l f'
  end
 end.
```

*Only one annotation needed!*

useless!

```
let rec lookup l f =
 match l with
 | Nil → assert false
 | Cons (a, l0) → (match f with
          | FinO _ → a
          | FinS (_, f') → lookup l0 f')
```

new!

```
let rec lookup l f =
 match l with
 | Nil → assert fals
 | Cons (a, l0) → (match f with
          | FinO → a
          | FinS f' → lookup l0 f')
```

# Isn’t It Just ℕ?

Before:

```
Inductive fin : ℕ → Set :=
| finO {n} : fin (S n)
| finS {n} : fin n → fin (S n).
```

```
type fin =
| FinO of nat
| FinS of nat * fin
```

# Isn’t It Just ℕ?

Before:

```
Inductive fin : ℕ → Set :=
| finO {n} : fin (S n)
| finS {n} : fin n → fin (S n).
```

```
type fin =
| FinO of
| FinS of      fin
```

bad!!
bad!!

After:

```
Inductive fin : ℕ@{Ghost} → Set :=
| finO {n} : fin (S n)
| finS {n} : fin n → fin (S n).
```

```
type fin =
| FinO
| FinS of fin
```

# Isn’t It Just ℕ?

Before:

```
Inductive fin : ℕ → Set :=
| finO {n} : fin (S n)
| finS {n} : fin n → fin (S n).
```

```
type fin =
| FinO of
| FinS of      fin
```

bad!!

bad!!

After:

```
Inductive fin : ℕ@{Ghost} → Set :=
| finO {n} : fin (S n)
| finS {n} : fin n → fin (S n).
```

```
type fin = nat?
```

# No Cheating Allowed!

```
Inductive vec (A : Type) : ℕ → Type :=
| VNil : vec A 0
| VCons : ∀ (n : ℕ), A → vec A n → vec A (S n).

Definition length {A n} (_ : vec A n) : ℕ := n.
```

```
type 'a vec =
| VNil
| VCons of nat * 'a * 'a vec

let length n _ = n
```

# No Cheating Allowed!

```
Inductive vec (A : Type) : ℕ → Type :=
| VNil : vec A 0
| VCons : ∀ (n : ℕ), A → vec A n → vec A (S n).

Definition length {A n} (_ : vec A n) : ℕ := n.
```

cheater!!

```
type 'a vec =
| VNil
| VCons of nat * 'a * 'a vec

let length n _ = n
```

Horrible extraction!

```
Inductive vec (A : Type) : ℕ@{Ghost} → Type :=
| VNil : vec A 0
| VCons : ∀ (n : ℕ@{Ghost}), A → vec A n → vec A (S n).

Fixpoint length {A n} (v : vec A n) : ℕ :=
 match v with
 | VNil ⇒ 0
 | VCons _ xs ⇒ S (length xs)
 end.
```

```
type 'a vec =
| VNil
| VCons of 'a * 'a vec

let rec length = function
| VNil → O
| VCons (_, xs) → S (length xs)
```

# No Cheating Allowed!

```
Inductive vec (A : Type) : ℕ → Type :=
| VNil : vec A 0
| VCons : ∀ (n : ℕ), A → vec A n → vec A (S n).

Definition length {A n} (_ : vec A n) : ℕ := n.
```

cheater!!

```
type 'a vec =
| VNil
| VCons of nat * 'a * 'a vec

let length n _ = n
```

Horrible extraction!

```
Inductive vec (A : Type) : ℕ@{Ghost} → Type :=
| VNil : vec A 0
| VCons : ∀ (n : ℕ@{Ghost}), A → vec A n → vec A (S n).

Fixpoint length {A n} (v : vec A n) : ℕ :=
 match v with
 | VNil ⇒ 0
 | VCons _ xs ⇒ S (length xs)
 end.
```

```
type 'a vec = 'a list?

let length = List.length?
```

# New Episode of *La Villa des Sorts*

Previously, in *La Villa des Sorts*...

Strict propositions ☹ `Acc` `eq`

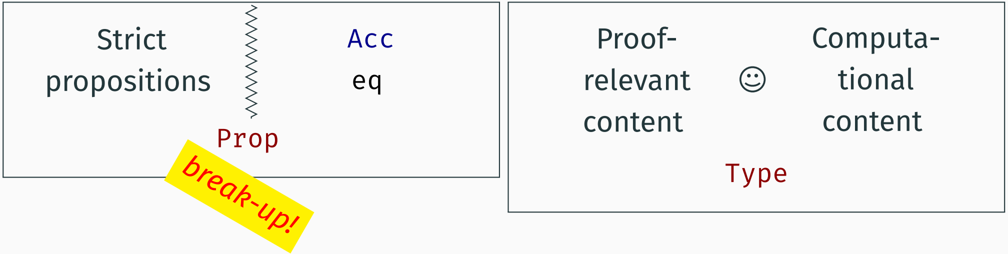
`Prop`

Proof-relevant content ☺ Computational content

`Type`

# New Episode of *La Villa des Sorts*

Previously, in *La Villa des Sorts*...



Strict propositions ≀ `Acc` `eq`

`Prop`

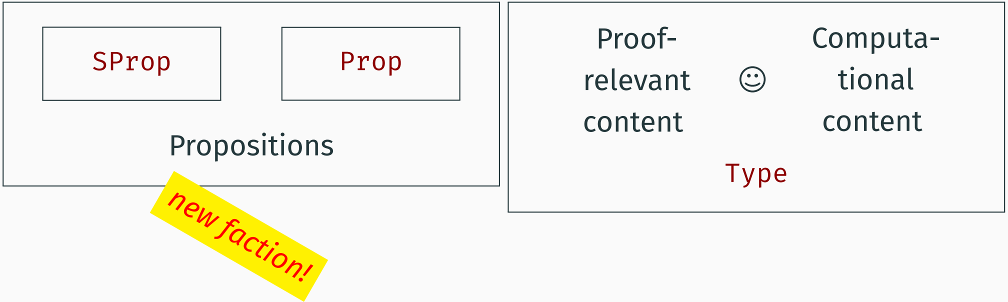
*break-up!*

Proof-relevant content ☺ Computational content

`Type`

# New Episode of *La Villa des Sorts*

Previously, in *La Villa des Sorts*...



`SProp`

`Prop`

Propositions

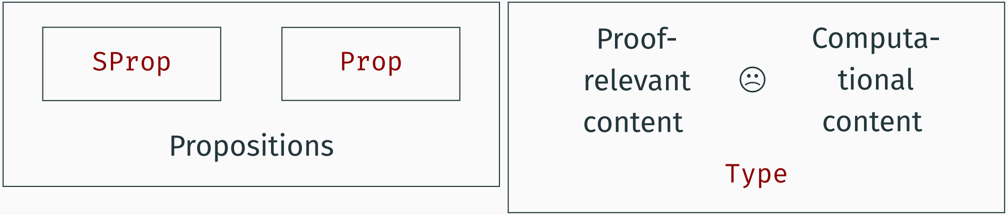
*new faction!*

Proof-relevant content ☺ Computational content

`Type`

# New Episode of *La Villa des Sorts*

Now...



`SProp`

`Prop`

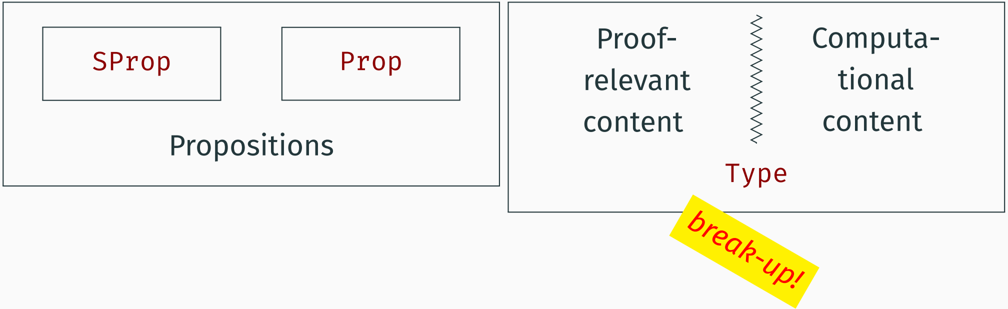
Propositions

Proof-relevant content ☹ Computational content

`Type`

# New Episode of *La Villa des Sorts*

Now...



SProp

Prop

Propositions

Proof-relevant content

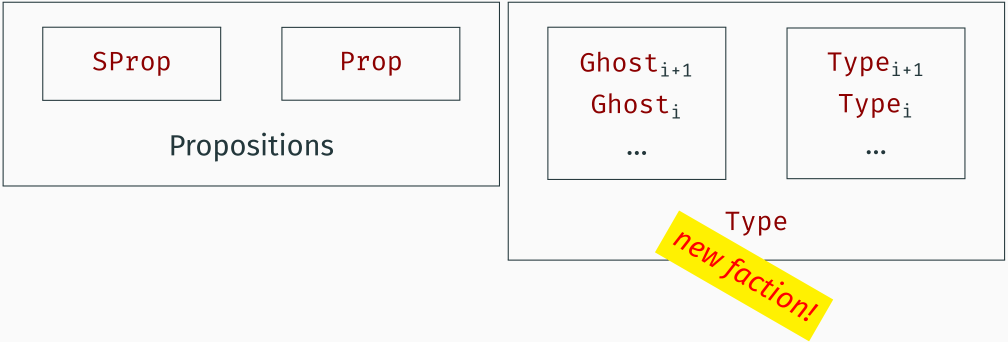
Computational content

Type

break-up!

# New Episode of *La Villa des Sorts*

Now...



SProp

Prop

Propositions

$\mathtt{Ghost}_{\mathtt{i+1}}$

$\mathtt{Ghost}_{\mathtt{i}}$

...

$\mathtt{Type}_{\mathtt{i+1}}$

$\mathtt{Type}_{\mathtt{i}}$

...

Type

*new faction!*

# You’re Stealing His Job!!

| Perks | s = Prop | s = Ghost |
|---|---|---|
| Erasure at extraction | ✓ | ✓ |
| Acc accomodation | ✓ | ✓ |
| Consistency | ✓ | $[\![\texttt{Ghost}_i]\!] = \texttt{Type}_i$ ✓ |
| true@{s} $\neq$ false@{s} | × | $\texttt{Ghost}_i : \texttt{Ghost}_{i+1}$ ✓ |
| Indices erasure | × | ✓ |
| Impredicativity | ✓ | × |

# The `Ghost` Sort

⚠ Heavy use of notions seen tuesday ahead!

Design #1:

- ⊥@{`Ghost`} elimination,
- redevelopment of logic (or
  wait for a sort poly. one).

# The `Ghost` Sort

⚠ Heavy use of notions seen tuesday ahead!

Design #1:

- $\bot$@{`Ghost`} elimination,
- redevelopment of logic (or wait for a sort poly. one).

Design #2:

- `Ghost` $\rightsquigarrow$ `SProp`,
- `SProp`'s logic.

# The Ghost Sort

⚠ Heavy use of notions seen tuesday ahead!

Design #1:

- ⊥@{Ghost} elimination,
- redevelopment of logic (or wait for a sort poly. one).

Design #2:

- Ghost $\rightsquigarrow$ SProp,
- SProp's logic.

Design #3:

- Ghost $\rightsquigarrow$ Prop,
- Prop's logic.

# Development Plan

The plan:

1. (More) Examples with designs.
2. (Current) MetaRocq Formalization.
3. Rocq Implementation.

Main MetaRocq goal — commutation of:

$$\begin{tikzcd}
t \arrow[r, "\text{evaluation}"] \arrow[d, "\text{erasure}"'] & v \arrow[d, "\text{erasure}"] \\
t' \arrow[r, "\text{evaluation}"'] & v'
\end{tikzcd}$$

# `Ghost` vs. QTT

| | QTT | `Ghost` |
|---|---|---|
| Addition | #(uses of a term) | Ground sort |
| Erasure | @0 t : A : Set | `t : A : Ghost` |
| Type-checking Changes | All rules | Eliminator rules |
| Erased Accessibility | No | Yes |

# Ghost vs. Erasure Modality

```
record Erased (@0 A : Set a) : Set a where
  constructor [_]
  field
    @0 erased : A
```

Key properties:

- Monad.
- Stable A := Erased A → A.
- DNE or dec. for A: Stable A.
- Revival: Erased A ≃ A.
- Few types can be revived.

# Ghost vs. Erasure Modality

```
record Erased (@0 A : Set a) : Set a where
  constructor [_]
  field
   @0 erased : A
```

In Rocq:

```
Inductive Box@{u} (A : Type@{u}) : Ghost@{u} :=
  box : A → Box A.
Inductive Unbox@{u} (A : Ghost@{u}) : Type@{u} :=
 unbox : A → Unbox A.

Definition Erased@{u} (A : Type@{u}) :=
 Unbox (Box A).
Notation "[ x ]" := (unbox (box x)).
```

Key properties:

- Monad.
- Stable A := Erased A → A.
- DNE or dec. for A: Stable A.
- Revival: Erased A ≃ A.
- Few types can be revived.

Ghost satisfy key properties with every design.

(But machinery level differs...)

# Conclusion

We have presented:

- Development plan.
- Current status.
- Comparison with existing systems.

Near future:

- Settle for a design.
- Preferred: $\texttt{Ghost} \rightsquigarrow \texttt{Prop}$.
- Allows: $\texttt{Ghost}_\texttt{i} \approx \texttt{Type}_\texttt{i}$ + computational sort.
- Focus on the formalization.

# Conclusion

We have presented:

- Development plan.
- Current status.
- Comparison with existing systems.

Near future:

- Settle for a design.
- Preferred: `Ghost` $\leadsto$ `Prop`.
- Allows: $\texttt{Ghost}_{\texttt{i}} \approx \texttt{Type}_{\texttt{i}}$ + computational sort.
- Focus on the formalization.

Thanks for your attention!