Definitional Proof Irrelevance Made Accessible

Thiago Felicissimo, Yann Leray, Loïc Pujet, Nicolas Tabareau, Éric Tanter & Théo Winterhalter

23 July 2026 @ LICS'26

INRIA

Prop = Type?

Prop = Type?

Propositions

Prop = Type?

Propositions Types

Prop = Type?

$$\text{Propositions} \quad \neq \quad \text{Types}$$

As in First-Order Logic (FOL), Higher-Order Logic (HOL), Isabelle

Prop = Type?

Propositions Types

Prop = Type?

$$\text{Propositions} \quad = \quad \text{Types}$$

As in Martin-Löf Type Theory (MLTT)

Prop = Type?

Propositions Types

Prop = Type?

$$\text{Propositions} \subset \text{Types}$$

As in Rocq, Agda, Lean: we have type universes Type and Prop

Prop = Type?

$$\text{Propositions} \quad \subset \quad \text{Types}$$

As in Rocq, Agda, Lean: we have type universes Type and Prop

Motto Distinguish *relevant* data of type A : Type from *irrelevant* proofs of P : Prop

If P : Prop and $p, q : P$ we might want $p = q$, but we always want true $\neq$ false

Ensuring compatibility of Prop with irrelevance

Constructing *relevant* data in Type from *irrelevant* proofs in Prop can be dangerous:

$$\left(\begin{array}{l}\lambda p.\ \text{match } p \text{ with} \\ \quad \mid \mathsf{inl}\ a \to \mathsf{true} \\ \quad \mid \mathsf{inr}\ b \to \mathsf{false}\end{array}\right) : A \vee B \to \mathsf{Bool}$$

Ensuring compatibility of Prop with irrelevance

Constructing *relevant* data in Type from *irrelevant* proofs in Prop can be dangerous:

$$\left(\begin{array}{l}\lambda p.\ \text{match } p \text{ with} \\ \quad \mid \mathsf{inl}\ a \to \mathsf{true} \\ \quad \mid \mathsf{inr}\ b \to \mathsf{false}\end{array}\right) \quad : A \vee B \to \mathsf{Bool}$$

If $\mathsf{inl}\ a = \mathsf{inr}\ b$ then $\mathsf{true} = \mathsf{false}$, yet we can prove that $\mathsf{true} \neq \mathsf{false}$...

Ensuring compatibility of Prop with irrelevance

Constructing *relevant* data in Type from *irrelevant* proofs in Prop can be dangerous:

$$\left(\begin{array}{l}\lambda p.\ \text{match } p \text{ with} \\ \quad\quad \mid \mathsf{inl}\ a \to \mathsf{true} \\ \quad\quad \mid \mathsf{inr}\ b \to \mathsf{false}\end{array}\right) \quad : A \vee B \to \mathsf{Bool}$$

If $\mathsf{inl}\ a = \mathsf{inr}\ b$ then $\mathsf{true} = \mathsf{false}$, yet we can prove that $\mathsf{true} \neq \mathsf{false}$...

In Rocq, *subsingleton criterion* allows to construct data in Type from three propositions:

Ensuring compatibility of Prop with irrelevance

Constructing *relevant* data in Type from *irrelevant* proofs in Prop can be dangerous:

$$\left(\begin{array}{l}\lambda p.\ \text{match } p \text{ with} \\ \quad\quad \mid \mathsf{inl}\ a \to \mathsf{true} \\ \quad\quad \mid \mathsf{inr}\ b \to \mathsf{false}\end{array}\right) \quad : A \vee B \to \mathsf{Bool}$$

If $\mathsf{inl}\ a = \mathsf{inr}\ b$ then $\mathsf{true} = \mathsf{false}$, yet we can prove that $\mathsf{true} \neq \mathsf{false}$...

In Rocq, *subsingleton criterion* allows to construct data in Type from three propositions:

1. the empty proposition False

Ensuring compatibility of Prop with irrelevance

Constructing *relevant* data in Type from *irrelevant* proofs in Prop can be dangerous:

$$\left(\begin{array}{l}\lambda p.\ \text{match } p \text{ with} \\ \quad \mid \mathsf{inl}\ a \to \mathsf{true} \\ \quad \mid \mathsf{inr}\ b \to \mathsf{false}\end{array}\right) \quad : A \vee B \to \mathsf{Bool}$$

If $\mathsf{inl}\ a = \mathsf{inr}\ b$ then $\mathsf{true} = \mathsf{false}$, yet we can prove that $\mathsf{true} \neq \mathsf{false}$...

In Rocq, *subsingleton criterion* allows to construct data in Type from three propositions:

1. the empty proposition False
2. the equality proposition $=$

Ensuring compatibility of Prop with irrelevance

Constructing *relevant* data in Type from *irrelevant* proofs in Prop can be dangerous:

$$\left(\begin{array}{l}\lambda p.\ \text{match } p \text{ with}\\ \quad \mid \mathsf{inl}\ a \to \mathsf{true}\\ \quad \mid \mathsf{inr}\ b \to \mathsf{false}\end{array}\right) \quad : A \vee B \to \mathsf{Bool}$$

If $\mathsf{inl}\ a = \mathsf{inr}\ b$ then $\mathsf{true} = \mathsf{false}$, yet we can prove that $\mathsf{true} \neq \mathsf{false}$...

In Rocq, *subsingleton criterion* allows to construct data in Type from three propositions:

1. the empty proposition False
2. the equality proposition $=$
3. the accessibility predicate Acc

Ensuring compatibility of Prop with irrelevance

Constructing *relevant* data in Type from *irrelevant* proofs in Prop can be dangerous:

$$\left(\begin{array}{l}\lambda p.\ \text{match } p \text{ with} \\ \quad \mid \mathsf{inl}\ a \to \mathsf{true} \\ \quad \mid \mathsf{inr}\ b \to \mathsf{false}\end{array}\right) \quad : A \vee B \to \mathsf{Bool}$$

If $\mathsf{inl}\ a = \mathsf{inr}\ b$ then $\mathsf{true} = \mathsf{false}$, yet we can prove that $\mathsf{true} \neq \mathsf{false}$...

In Rocq, *subsingleton criterion* allows to construct data in Type from three propositions:

1. the empty proposition False
2. the equality proposition $=$
3. the accessibility predicate Acc

Ensures consistency of Prop with proof-irrel : $\forall (P : \mathsf{Prop})\ (p\, q : P).\ p = q$

Ensuring compatibility of Prop with irrelevance

Constructing *relevant* data in Type from *irrelevant* proofs in Prop can be dangerous:

$$\left(\begin{array}{l}\lambda p.\ \text{match } p \text{ with}\\ \quad \mid \mathsf{inl}\ a \to \mathsf{true}\\ \quad \mid \mathsf{inr}\ b \to \mathsf{false}\end{array}\right) \quad : A \vee B \to \mathsf{Bool}$$

If $\mathsf{inl}\ a = \mathsf{inr}\ b$ then $\mathsf{true} = \mathsf{false}$, yet we can prove that $\mathsf{true} \neq \mathsf{false}$...

In Rocq, *subsingleton criterion* allows to construct data in Type from three propositions:

1. the empty proposition False
2. the equality proposition $=$
3. the accessibility predicate Acc

Ensures consistency of Prop with proof-irrel : $\forall (P : \mathsf{Prop})\ (p\,q : P).\ p = q$

Enables *extraction* into common programming languages (eg OCaml)

Accessibility: Positive well-foundedness

$$\frac{R : A \to A \to \mathsf{Prop} \qquad a : A}{\mathsf{Acc}\ R\ a : \mathsf{Prop}}$$

Accessibility: Positive well-foundedness

$$\frac{R : A \to A \to \mathsf{Prop} \qquad a : A}{\mathsf{Acc}\ R\ a : \mathsf{Prop}} \qquad \frac{p : \forall (b : A).\ R\ b\ a \to \mathsf{Acc}\ R\ b}{\mathsf{acc\text{-}in}\ p : \mathsf{Acc}\ R\ a}$$

Accessibility: Positive well-foundedness

$$\frac{R : A \to A \to \mathsf{Prop} \qquad a : A}{\mathsf{Acc}\ R\ a : \mathsf{Prop}} \qquad \frac{p : \forall (b : A).\ R\ b\ a \to \mathsf{Acc}\ R\ b}{\mathsf{acc\text{-}in}\ p : \mathsf{Acc}\ R\ a}$$

$$\frac{P : A \to \mathsf{Type} \qquad p : \forall (a : A)\ (rec : \forall (b : A).\ R\ b\ a \to P\ b).\ P\ a}{\mathsf{acc\text{-}el}\ P\ p : \forall (a : A).\, \mathsf{Acc}\ R\ a \to P\ a}$$

Accessibility: Positive well-foundedness

$$\frac{R : A \to A \to \mathsf{Prop} \qquad a : A}{\mathsf{Acc}\ R\ a : \mathsf{Prop}} \qquad \frac{p : \forall (b : A).\ R\ b\ a \to \mathsf{Acc}\ R\ b}{\mathsf{acc\text{-}in}\ p : \mathsf{Acc}\ R\ a}$$

$$\frac{P : A \to \mathsf{Type} \qquad p : \forall (a : A)\ (rec : \forall (b : A).\ R\ b\ a \to P\ b).\ P\ a}{\mathsf{acc\text{-}el}\ P\ p : \forall (a : A).\, \mathsf{Acc}\ R\ a \to P\ a}$$

Used in proof assistants to elaborate definitions by well-founded recursion:

```
Equations? gcd (x y : nat) : nat by wf (x + y) lt :=
gcd 0 x := x ;       gcd x 0 := x ;
gcd x y with gt_eq_gt_dec x y := {
| inleft (left _) := gcd x (y - x) ;
| inleft (right refl) := x ;
| inright _ := gcd (x - y) y }.
Proof. all: lia. Defined.
```

Propositional irrelevance versus definitional irrelevance

Propositional proof irrelevance If $P : \mathsf{Prop}$ and $p, q : P$, then there is $e : p = q$

Any two $(n, p), (n, q)$ of type $\Sigma(x : \mathsf{Nat}).\, x > 0$ are always propositionally equal

But might not be *convertible*: the user needs to manually transport terms

Propositional irrelevance versus definitional irrelevance

Propositional proof irrelevance If $P : \mathsf{Prop}$ and $p, q : P$, then there is $e : p = q$

Any two $(n, p), (n, q)$ of type $\Sigma(x : \mathsf{Nat}).\, x > 0$ are always propositionally equal

But might not be *convertible*: the user needs to manually transport terms

Definitional **proof irrelevance** If $P : \mathsf{Prop}$ and $p, q : P$, then $p \equiv q$ (p and q are *convertible*)

Now, $(n, p), (n, q)$ are always interchangeable, closer to mathematical practice

Integrated into type-theoretic proof assistants (Agda, Rocq & Lean) in universe SProp

Failure of the subsingleton criterion

Unfortunately, subsingleton criterion interacts badly with SProp...

Failure of the subsingleton criterion

Unfortunately, subsingleton criterion interacts badly with SProp...

Eliminating from SProp to Type with

Failure of the subsingleton criterion

Unfortunately, subsingleton criterion interacts badly with SProp...

Eliminating from SProp to Type with

- False: Still ok (even in presence of consistent axioms!)

Failure of the subsingleton criterion

Unfortunately, subsingleton criterion interacts badly with SProp...

Eliminating from SProp to Type with

- False: Still ok (even in presence of consistent axioms!)
- Equality (=): In presence of impredicativity, breaks proof normalization

Failure of the subsingleton criterion

Unfortunately, subsingleton criterion interacts badly with SProp...

Eliminating from SProp to Type with

- False: Still ok (even in presence of consistent axioms!)
- Equality ($=$): In presence of impredicativity, breaks proof normalization
  As with Prop, in presence of *function extensionality*, breaks *canonicity*, that is:

$$\vdash t : \mathsf{Nat} \text{ implies } \vdash t \equiv \mathsf{S}^n(\mathsf{0}) : \mathsf{Nat} \text{ for some } n$$

Failure of the subsingleton criterion

Unfortunately, subsingleton criterion interacts badly with SProp...

Eliminating from SProp to Type with

- False: Still ok (even in presence of consistent axioms!)
- Equality (=): In presence of impredicativity, breaks proof normalization
  As with Prop, in presence of *function extensionality*, breaks *canonicity*, that is:

  $$\vdash t : \mathsf{Nat} \text{ implies } \vdash t \equiv \mathsf{S}^n(\mathsf{0}) : \mathsf{Nat} \text{ for some } n$$

- Acc: Breaks decidability of conversion/type checking (= proof checking)

Failure of the subsingleton criterion

Unfortunately, subsingleton criterion interacts badly with SProp...

Eliminating from SProp to Type with

- False: Still ok (even in presence of consistent axioms!)
- Equality (=): In presence of impredicativity, breaks proof normalization
  As with Prop, in presence of *function extensionality*, breaks *canonicity*, that is:

  $$\vdash t : \mathsf{Nat} \text{ implies } \vdash t \equiv \mathsf{S}^n(\mathsf{0}) : \mathsf{Nat} \text{ for some } n$$

- Acc: Breaks decidability of conversion/type checking (= proof checking)
  Intuition: By definitional proof irrel, *any* $p : \mathsf{Acc}\ R\ a$ is convertible to some $\mathsf{accin}\ q$
  Thus any acc-el must be evaluated, even if p is "false", such as $p : \mathsf{Acc}\ (\lambda xy.\mathsf{True})\ 0$

Dealing with SProp in proof assistants

Agda, Rocq: Only False can be eliminated into Type

Preserves good properties of the theory, but highly limits applicability

In practice, users still need to resort to Prop...

Dealing with SProp in proof assistants

Agda, Rocq: Only False can be eliminated into Type

Preserves good properties of the theory, but highly limits applicability

In practice, users still need to resort to Prop...

Lean: Implements full subsingleton criterion, and lives with its consequences

No canonicity, because (1) elimination of equality into Type, and (2) Hilbert's ε operator

Undecidable type-checking, implementation less stable

Since v4.19.0, undecidability hidden by elaboration when defining functions by well-founded rec.

Dealing with SProp in proof assistants

Agda, Rocq: Only False can be eliminated into Type

Preserves good properties of the theory, but highly limits applicability

In practice, users still need to resort to Prop...

Lean: Implements full subsingleton criterion, and lives with its consequences

No canonicity, because (1) elimination of equality into Type, and (2) Hilbert's ε operator

Undecidable type-checking, implementation less stable

Since v4.19.0, undecidability hidden by elaboration when defining functions by well-founded rec.

Our goal A design combining SProp with equality and Acc, but preserving good properties

Reconciling $=$ with SProp, with *observational equality*

As remarked by Pujet & Tabareau, problems of SProp with $=$ solved by McBride's & Altenkirch's *observational equality*

Reconciling $=$ with SProp, with *observational equality*

As remarked by Pujet & Tabareau, problems of SProp with $=$ solved by McBride's & Altenkirch's *observational equality*

Instead of Martin-Löf's **J** eliminator, eliminated using a *cast* operator:

$$\frac{A, B : \mathsf{Type} \qquad p : A = B \qquad a : A}{\mathsf{cast}_p^{A \rightsquigarrow B}(a) : B}$$

Reconciling $=$ with SProp, with *observational equality*

As remarked by Pujet & Tabareau, problems of SProp with $=$ solved by McBride's & Altenkirch's *observational equality*

Instead of Martin-Löf's **J** eliminator, eliminated using a *cast* operator:

$$\frac{A, B : \mathsf{Type} \qquad p : A = B \qquad a : A}{\mathsf{cast}_p^{A \rightsquigarrow B}(a) : B}$$

Crucial property Unlike **J**, cast computes by case analysis on types:

$$\mathsf{cast}_p^{(A \times B) \rightsquigarrow (A' \times B')} t \longrightarrow \langle \mathsf{cast}_{p.1}^{A \rightsquigarrow A'}(\pi_1 t), \mathsf{cast}_{p.2}^{B \rightsquigarrow B'}(\pi_2 t) \rangle$$

Reconciling $=$ with SProp, with *observational equality*

As remarked by Pujet & Tabareau, problems of SProp with $=$ solved by McBride's & Altenkirch's *observational equality*

Instead of Martin-Löf's **J** eliminator, eliminated using a *cast* operator:

$$\frac{A, B : \mathsf{Type} \qquad p : A = B \qquad a : A}{\mathsf{cast}_p^{A \rightsquigarrow B}(a) : B}$$

Crucial property Unlike **J**, cast computes by case analysis on types:

$$\mathsf{cast}_p^{(A \times B) \rightsquigarrow (A' \times B')} t \longrightarrow \langle \mathsf{cast}_{p.1}^{A \rightsquigarrow A'}(\pi_1 t), \mathsf{cast}_{p.2}^{B \rightsquigarrow B'}(\pi_2 t) \rangle$$

Rules *never look inside equality proofs*, hence equality axioms cannot block reduction!

Reconciling $=$ with SProp, with *observational equality*

As remarked by Pujet & Tabareau, problems of SProp with $=$ solved by McBride's & Altenkirch's *observational equality*

Instead of Martin-Löf's **J** eliminator, eliminated using a *cast* operator:

$$\frac{A, B : \mathsf{Type} \qquad p : A = B \qquad a : A}{\mathsf{cast}_p^{A \rightsquigarrow B}(a) : B}$$

Crucial property Unlike **J**, cast computes by case analysis on types:

$$\mathsf{cast}_p^{(A \times B) \rightsquigarrow (A' \times B')} t \longrightarrow \langle \mathsf{cast}_{p.1}^{A \rightsquigarrow A'}(\pi_1 t), \mathsf{cast}_{p.2}^{B \rightsquigarrow B'}(\pi_2 t) \rangle$$

Rules *never look inside equality proofs*, hence equality axioms cannot block reduction!

Combined with SProp yields $\mathsf{CC}^{\mathsf{obs}}$, well-behaved theory for set-level mathematics

Enjoys canonicity, decidability of typing and consistency, as shown by Pujet & Tabareau

Usual **J** eliminator can still be simulated in the theory

This work: Reconciling Acc with SProp

We propose a design combining two theories that extend $\mathsf{CC}^{\mathsf{obs}}$ with Acc:

This work: Reconciling Acc with SProp

We propose a design combining two theories that extend $\mathsf{CC}^{\mathsf{obs}}$ with Acc:

$$\mathcal{T}^{=}_{\mathrm{Acc}}$$

Eliminator acc-el computes up to $=$

This work: Reconciling Acc with SProp

We propose a design combining two theories that extend $\mathsf{CC}^{\mathsf{obs}}$ with Acc:

$$\mathcal{T}^{=}_{\mathrm{Acc}}$$

Eliminator acc-el computes up to $=$

Fact $\mathcal{T}^{=}_{\mathrm{Acc}}$ has decidable type-checking
Extends $\mathsf{CC}^{\mathsf{obs}}$ with just constants

This work: Reconciling Acc with SProp

We propose a design combining two theories that extend $\mathsf{CC}^{\mathsf{obs}}$ with Acc:

$$\mathcal{T}^{=}_{\mathrm{Acc}}$$

Eliminator acc-el computes up to $=$

Fact $\mathcal{T}^{=}_{\mathrm{Acc}}$ has decidable type-checking
Extends $\mathsf{CC}^{\mathsf{obs}}$ with just constants

But usual form of canonicity fails...

$\mathsf{gcd}\ 8\ 2 = 2$ does not follow from conversion

This work: Reconciling Acc with SProp

We propose a design combining two theories that extend $\mathsf{CC}^{\mathsf{obs}}$ with Acc:

$$\mathcal{T}^{=}_{\mathrm{Acc}} \quad \subset \quad \mathcal{T}^{\equiv}_{\mathrm{Acc}}$$

Eliminator acc-el computes up to $=$

Fact $\mathcal{T}^{=}_{\mathrm{Acc}}$ has decidable type-checking
Extends $\mathsf{CC}^{\mathsf{obs}}$ with just constants

But usual form of canonicity fails...

$\mathsf{gcd}\ 8\ 2 = 2$ does not follow from conversion

Eliminator acc-el computes definitionally
Hence conversion/typing are undecidable

This work: Reconciling Acc with SProp

We propose a design combining two theories that extend $\mathsf{CC}^{\mathsf{obs}}$ with Acc:

$$\mathcal{T}^{=}_{\mathrm{Acc}} \quad \subset \quad \mathcal{T}^{\equiv}_{\mathrm{Acc}}$$

Eliminator acc-el computes up to $=$

Fact $\mathcal{T}^{=}_{\mathrm{Acc}}$ has decidable type-checking
Extends $\mathsf{CC}^{\mathsf{obs}}$ with just constants

But usual form of canonicity fails...

Eliminator acc-el computes definitionally
Hence conversion/typing are undecidable

Thm $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ satisfies canonicity

$\mathsf{gcd}\ 8\ 2 = 2$ follows from conversion

This work: Reconciling Acc with SProp

We propose a design combining two theories that extend $\mathsf{CC}^{\mathsf{obs}}$ with Acc:

Eliminator acc-el computes up to $=$

Fact $\mathcal{T}^{=}_{\mathrm{Acc}}$ has decidable type-checking
Extends $\mathsf{CC}^{\mathsf{obs}}$ with just constants

But usual form of canonicity fails...

Eliminator acc-el computes definitionally
Hence conversion/typing are undecidable

Thm $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ satisfies canonicity

$\mathsf{gcd}\ 8\ 2 = 2$ follows from conversion

Thm $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ is conservative over $\mathcal{T}^{=}_{\mathrm{Acc}}$:
If $\vdash^{=} P : \mathsf{SProp}$ and $\vdash^{\equiv} p : P$
then $\vdash^{=} q : P$ for some q

This work: Reconciling Acc with SProp

We propose a design combining two theories that extend $\mathsf{CC}^{\mathsf{obs}}$ with Acc:

Eliminator acc-el computes up to $=$

Fact $\mathcal{T}^{=}_{\mathrm{Acc}}$ has decidable type-checking
Extends $\mathsf{CC}^{\mathsf{obs}}$ with just constants

Cor $\mathcal{T}^{=}_{\mathrm{Acc}}$ enjoys *propositional* canonicity:
If $\vdash t : \mathsf{Nat}$ then $\vdash e : t = \mathsf{S}^n(\mathsf{0})$ for some e, n

Eliminator acc-el computes definitionally
Hence conversion/typing are undecidable

Thm $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ satisfies canonicity

$\mathsf{gcd}\ 8\ 2 = 2$ follows from conversion

Thm $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ is conservative over $\mathcal{T}^{=}_{\mathrm{Acc}}$:
If $\vdash^{=} P : \mathsf{SProp}$ and $\vdash^{\equiv} p : P$
then $\vdash^{=} q : P$ for some q

This work: Reconciling Acc with SProp

We propose a design combining two theories that extend $\mathsf{CC}^{\mathsf{obs}}$ with Acc:

Eliminator acc-el computes up to $=$

Fact $\mathcal{T}^{=}_{\mathrm{Acc}}$ has decidable type-checking
Extends $\mathsf{CC}^{\mathsf{obs}}$ with just constants

Cor $\mathcal{T}^{=}_{\mathrm{Acc}}$ enjoys *propositional* canonicity:
If $\vdash t : \mathsf{Nat}$ then $\vdash e : t = \mathsf{S}^n(\mathsf{0})$ for some e, n

Eliminator acc-el computes definitionally
Hence conversion/typing are undecidable

Thm $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ satisfies canonicity

$\mathsf{gcd}\ 8\ 2 = 2$ follows from conversion

Thm $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ is conservative over $\mathcal{T}^{=}_{\mathrm{Acc}}$:
If $\vdash^{=} P : \mathsf{SProp}$ and $\vdash^{\equiv} p : P$
then $\vdash^{=} q : P$ for some q

Thm $\mathcal{T}^{=}_{\mathrm{Acc}}$ and $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ admit set-theoretic models, in particular they are consistent

This work: Reconciling Acc with SProp

We propose a design combining two theories that extend $\mathsf{CC}^{\mathsf{obs}}$ with Acc:

Eliminator acc-el computes up to $=$

Fact $\mathcal{T}^{=}_{\mathrm{Acc}}$ has decidable type-checking
Extends $\mathsf{CC}^{\mathsf{obs}}$ with just constants

Cor $\mathcal{T}^{=}_{\mathrm{Acc}}$ enjoys *propositional* canonicity:
If $\vdash t : \mathsf{Nat}$ then $\vdash e : t = \mathsf{S}^n(\mathsf{0})$ for some e, n

Eliminator acc-el computes definitionally
Hence conversion/typing are undecidable

Thm $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ satisfies canonicity

$\mathsf{gcd}\ 8\ 2 = 2$ follows from conversion

Thm $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ is conservative over $\mathcal{T}^{=}_{\mathrm{Acc}}$:
If $\vdash^{=} P : \mathsf{SProp}$ and $\vdash^{\equiv} p : P$
then $\vdash^{=} q : P$ for some q

Thm $\mathcal{T}^{=}_{\mathrm{Acc}}$ and $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ admit set-theoretic models, in particular they are consistent

Results formalized in Rocq

The design in practice

$\mathcal{T}^{=}_{\mathrm{Acc}}$ implemented on top of observational fork of RocQ of Pujet, Leray & Tabareau

Proof mode switch to $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ to ease proofs, justified by conservativity theorem

The design in practice

$\mathcal{T}^{=}_{\mathrm{Acc}}$ implemented on top of observational fork of Rocq of Pujet, Leray & Tabareau

Proof mode switch to $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ to ease proofs, justified by conservativity theorem

In $\mathcal{T}^{=}_{\mathrm{Acc}}$:

```
Lemma gcd_test : (gcd (2 ^ N) 2 <? 5) ~ true.
Proof.
  auto_Acc_unfold; reflexivity.
Qed.
```

The design in practice

$\mathcal{T}^{=}_{\mathrm{Acc}}$ implemented on top of observational fork of Rocq of Pujet, Leray & Tabareau

Proof mode switch to $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ to ease proofs, justified by conservativity theorem

In $\mathcal{T}^{=}_{\mathrm{Acc}}$:

```
Lemma gcd_test : (gcd (2 ^ N) 2 <? 5) ~ true.
Proof.
  auto_Acc_unfold; reflexivity.
Qed.
```

In $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$:

```
#[rewrite_rules(Acc_el_def)]
Lemma gcd_test_def (gcd (2 ^ N) 2 <? 5) ~ true.
Proof.
  reflexivity.
Qed.
```

The design in practice

$\mathcal{T}^{=}_{\mathrm{Acc}}$ implemented on top of observational fork of Rocq of Pujet, Leray & Tabareau

Proof mode switch to $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ to ease proofs, justified by conservativity theorem

In $\mathcal{T}^{=}_{\mathrm{Acc}}$:

```
Lemma gcd_test : (gcd (2 ^ N) 2 <? 5) ~ true.
Proof.
  auto_Acc_unfold; reflexivity.
Qed.
```

In $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$:

```
#[rewrite_rules(Acc_el_def)]
Lemma gcd_test_def (gcd (2 ^ N) 2 <? 5) ~ true.
Proof.
  reflexivity.
Qed.
```

For $N = 10$, takes 0.004 s

The design in practice

$\mathcal{T}^{=}_{\mathrm{Acc}}$ implemented on top of observational fork of Rocq of Pujet, Leray & Tabareau

Proof mode switch to $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ to ease proofs, justified by conservativity theorem

In $\mathcal{T}^{=}_{\mathrm{Acc}}$:

```
Lemma gcd_test : (gcd (2 ^ N) 2 <? 5) ~ true.
Proof.
  auto_Acc_unfold; reflexivity.
Qed.
```

For $N = 10$, timeouts after 10 min…

In $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$:

```
#[rewrite_rules(Acc_el_def)]
Lemma gcd_test_def (gcd (2 ^ N) 2 <? 5) ~ true.
Proof.
  reflexivity.
Qed.
```

For $N = 10$, takes 0.004 s

The design in practice

$\mathcal{T}^{=}_{\mathrm{Acc}}$ implemented on top of observational fork of Rocq of Pujet, Leray & Tabareau

Proof mode switch to $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ to ease proofs, justified by conservativity theorem

In $\mathcal{T}^{=}_{\mathrm{Acc}}$:

```
Lemma gcd_test : (gcd (2 ^ N) 2 <? 5) ~ true.
Proof.
  auto_Acc_unfold; reflexivity.
Qed.
```

For $N = 10$, timeouts after 10 min...

In $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$:

```
#[rewrite_rules(Acc_el_def)]
Lemma gcd_test_def (gcd (2 ^ N) 2 <? 5) ~ true.
Proof.
  reflexivity.
Qed.
```

For $N = 10$, takes 0.004 s

Crucial remark No need to implement costly translation from $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ to $\mathcal{T}^{=}_{\mathrm{Acc}}$:
Proofs are irrelevant, so can be treated as opaque!

Canonicity of $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$

We prove canonicity of $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ by adapting logical relation of Abel et al

Canonicity of $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$

We prove canonicity of $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ by adapting logical relation of Abel et al

We define reducibility relation

$$\Vdash t : A$$

stating that t evaluates to canonical element of A (for positive types)

Canonicity of $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$

We prove canonicity of $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ by adapting logical relation of Abel et al

We define reducibility relation

$$\Vdash t : A$$

stating that t evaluates to canonical element of A (for positive types)

Remark Because normalization does not hold, $\Vdash$ defined only on closed terms, unlike Abel et al

Canonicity of $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$

We prove canonicity of $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ by adapting logical relation of Abel et al

We define reducibility relation

$$\Vdash t : A$$

stating that t evaluates to canonical element of A (for positive types)

Remark Because normalization does not hold, $\Vdash$ defined only on closed terms, unlike Abel et al

Problem As shown by Coquand and Abel, proof normalization does not hold

Impredicativity + elimination of SProp equality into Type yields non-terminating proofs

Canonicity of $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$

We prove canonicity of $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ by adapting logical relation of Abel et al

We define reducibility relation

$$\Vdash t : A$$

stating that t evaluates to canonical element of A (for positive types)

Remark Because normalization does not hold, $\Vdash$ defined only on closed terms, unlike Abel et al

Problem As shown by Coquand and Abel, proof normalization does not hold

Impredicativity + elimination of SProp equality into Type yields non-terminating proofs

Counter-example reproducible even on closed terms using proposition extensionality

Canonicity of $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$

We prove canonicity of $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ by adapting logical relation of Abel et al

We define reducibility relation

$$\Vdash t : A$$

stating that t evaluates to canonical element of A (for positive types)

Remark Because normalization does not hold, $\Vdash$ defined only on closed terms, unlike Abel et al

Problem As shown by Coquand and Abel, proof normalization does not hold

Impredicativity + elimination of SProp equality into Type yields non-terminating proofs

Counter-example reproducible even on closed terms using proposition extensionality

Solution Like in previous work on $\mathsf{CC}^{\mathsf{obs}}$, logical relation at SProp is trivial:

$$\Vdash p : P : \mathsf{SProp} \quad := \quad \vdash p : P : \mathsf{SProp}$$

Fundamental theorem of logical relation

Canonicity for $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ follows from *fundamental theorem*, stating that logical relation is a model:

$$\Gamma \vdash t : A \quad \Rightarrow \quad \Vdash t[\sigma] : A[\sigma] \text{ for all } \Vdash \sigma : \Gamma$$

$$\Gamma \vdash t \equiv u : A \quad \Rightarrow \quad \Vdash t[\sigma] \equiv u[\sigma] : A[\sigma] \text{ for all } \Vdash \sigma : \Gamma$$

Fundamental theorem of logical relation

Canonicity for $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ follows from *fundamental theorem*, stating that logical relation is a model:

$$\Gamma \vdash t : A \quad \Rightarrow \quad \Vdash t[\sigma] : A[\sigma] \text{ for all } \Vdash \sigma : \Gamma$$

$$\Gamma \vdash t \equiv u : A \quad \Rightarrow \quad \Vdash t[\sigma] \equiv u[\sigma] : A[\sigma] \text{ for all } \Vdash \sigma : \Gamma$$

Main part of the proof is to show reducibility of eliminators for positive types

Fundamental theorem of logical relation

Canonicity for $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ follows from *fundamental theorem*, stating that logical relation is a model:

$$\Gamma \vdash t : A \quad \Rightarrow \quad \Vdash t[\sigma] : A[\sigma] \text{ for all } \Vdash \sigma : \Gamma$$

$$\Gamma \vdash t \equiv u : A \quad \Rightarrow \quad \Vdash t[\sigma] \equiv u[\sigma] : A[\sigma] \text{ for all } \Vdash \sigma : \Gamma$$

Main part of the proof is to show reducibility of eliminators for positive types

Example If $\Vdash t : \mathsf{Nat}$, we prove $\Vdash \mathsf{nat\text{-}rec}\ P\ p_0\ p_\mathsf{S}\ t : P\ t$ using that t evaluates to some $\mathsf{S}^n(\mathsf{0})$

Fundamental theorem of logical relation

Canonicity for $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ follows from *fundamental theorem*, stating that logical relation is a model:

$$\Gamma \vdash t : A \quad \Rightarrow \quad \Vdash t[\sigma] : A[\sigma] \text{ for all } \Vdash \sigma : \Gamma$$

$$\Gamma \vdash t \equiv u : A \quad \Rightarrow \quad \Vdash t[\sigma] \equiv u[\sigma] : A[\sigma] \text{ for all } \Vdash \sigma : \Gamma$$

Main part of the proof is to show reducibility of eliminators for positive types

Example If $\Vdash t : \mathsf{Nat}$, we prove $\Vdash \mathsf{nat\text{-}rec}\ P\ p_0\ p_\mathsf{S}\ t : P\ t$ using that t evaluates to some $\mathsf{S}^n(\mathsf{0})$

Problem $\Vdash q : \mathsf{Acc}\ R\ a$ gives no information, how to prove $\Vdash \mathsf{acc\text{-}el}\ P\ p\ a\ q : P\ a$?

Fundamental theorem of logical relation

Canonicity for $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ follows from *fundamental theorem*, stating that logical relation is a model:

$$\Gamma \vdash t : A \quad \Rightarrow \quad \Vdash t[\sigma] : A[\sigma] \text{ for all } \Vdash \sigma : \Gamma$$

$$\Gamma \vdash t \equiv u : A \quad \Rightarrow \quad \Vdash t[\sigma] \equiv u[\sigma] : A[\sigma] \text{ for all } \Vdash \sigma : \Gamma$$

Main part of the proof is to show reducibility of eliminators for positive types

Example If $\Vdash t : \mathsf{Nat}$, we prove $\Vdash \mathsf{nat\text{-}rec}\ P\ p_0\ p_\mathsf{S}\ t : P\ t$ using that t evaluates to some $\mathsf{S}^n(0)$

Problem $\Vdash q : \mathsf{Acc}\ R\ a$ gives no information, how to prove $\Vdash \mathsf{acc\text{-}el}\ P\ p\ a\ q : P\ a$?

Insight By standard set model, $\vdash q : \mathsf{Acc}\ R\ a$ gives proof that $[\![a]\!]$ is (meta-)accessible for $[\![R]\!]$

Fundamental theorem of logical relation

Canonicity for $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ follows from *fundamental theorem*, stating that logical relation is a model:

$$\Gamma \vdash t : A \quad \Rightarrow \quad \Vdash t[\sigma] : A[\sigma] \text{ for all } \Vdash \sigma : \Gamma$$

$$\Gamma \vdash t \equiv u : A \quad \Rightarrow \quad \Vdash t[\sigma] \equiv u[\sigma] : A[\sigma] \text{ for all } \Vdash \sigma : \Gamma$$

Main part of the proof is to show reducibility of eliminators for positive types

Example If $\Vdash t : \mathsf{Nat}$, we prove $\Vdash \mathsf{nat\text{-}rec}\ P\ p_0\ p_{\mathsf{S}}\ t : P\ t$ using that t evaluates to some $\mathsf{S}^n(\mathsf{0})$

Problem $\Vdash q : \mathsf{Acc}\ R\ a$ gives no information, how to prove $\Vdash \mathsf{acc\text{-}el}\ P\ p\ a\ q : P\ a$?

Insight By standard set model, $\vdash q : \mathsf{Acc}\ R\ a$ gives proof that $[\![a]\!]$ is (meta-)accessible for $[\![R]\!]$

It follows that a is (meta-)accessible for $\widetilde{R}(a, b) := \exists r. \vdash r : R\ a\ b$

Fundamental theorem of logical relation

Canonicity for $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ follows from *fundamental theorem*, stating that logical relation is a model:

$$\Gamma \vdash t : A \quad \Rightarrow \quad \Vdash t[\sigma] : A[\sigma] \text{ for all } \Vdash \sigma : \Gamma$$

$$\Gamma \vdash t \equiv u : A \quad \Rightarrow \quad \Vdash t[\sigma] \equiv u[\sigma] : A[\sigma] \text{ for all } \Vdash \sigma : \Gamma$$

Main part of the proof is to show reducibility of eliminators for positive types

Example If $\Vdash t : \mathsf{Nat}$, we prove $\Vdash \mathsf{nat\text{-}rec}\ P\ p_0\ p_\mathsf{S}\ t : P\ t$ using that t evaluates to some $\mathsf{S}^n(\mathsf{0})$

Problem $\Vdash q : \mathsf{Acc}\ R\ a$ gives no information, how to prove $\Vdash \mathsf{acc\text{-}el}\ P\ p\ a\ q : P\ a$?

Insight By standard set model, $\vdash q : \mathsf{Acc}\ R\ a$ gives proof that $[\![a]\!]$ is (meta-)accessible for $[\![R]\!]$

It follows that a is (meta-)accessible for $\widetilde{R}(a, b) := \exists r. \vdash r : R\ a\ b$

Solution We prove $\Vdash \mathsf{acc\text{-}el}\ P\ p\ a\ q : P\ a$ by induction on (meta-)accessibility proof of a for $\widetilde{R}$

Fundamental theorem of logical relation

Canonicity for $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ follows from *fundamental theorem*, stating that logical relation is a model:

$$\Gamma \vdash t : A \quad \Rightarrow \quad \Vdash t[\sigma] : A[\sigma] \text{ for all } \Vdash \sigma : \Gamma$$

$$\Gamma \vdash t \equiv u : A \quad \Rightarrow \quad \Vdash t[\sigma] \equiv u[\sigma] : A[\sigma] \text{ for all } \Vdash \sigma : \Gamma$$

Main part of the proof is to show reducibility of eliminators for positive types

Example If $\Vdash t : \mathsf{Nat}$, we prove $\Vdash \mathsf{nat\text{-}rec}\ P\ p_0\ p_\mathsf{S}\ t : P\ t$ using that t evaluates to some $\mathsf{S}^n(\mathsf{0})$

Problem $\Vdash q : \mathsf{Acc}\ R\ a$ gives no information, how to prove $\Vdash \mathsf{acc\text{-}el}\ P\ p\ a\ q : P\ a$?

Insight By standard set model, $\vdash q : \mathsf{Acc}\ R\ a$ gives proof that $[\![a]\!]$ is (meta-)accessible for $[\![R]\!]$

It follows that a is (meta-)accessible for $\widetilde{R}(a, b) := \exists r. \vdash r : R\ a\ b$

Solution We prove $\Vdash \mathsf{acc\text{-}el}\ P\ p\ a\ q : P\ a$ by induction on (meta-)accessibility proof of a for $\widetilde{R}$

Remark Proof holds in presence of any axioms validated by standard set model

We can use classical principles without jeopardizing canonicity!

Conclusion

We propose a design combining two theories with SProp, observational equality and Acc:

Eliminator acc-el computes up to $=$

- Has decidable type-checking
- Enjoys propositional canonicity

Eliminator acc-el computes definitionally

- Enjoys canonicity
- Conservative over $\mathcal{T}^{=}_{\mathrm{Acc}}$

Both $\mathcal{T}^{=}_{\mathrm{Acc}}$ and $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ admit set-theoretic models, and are in particular consistent

Results formalized in Rocq: `https://github.com/thiagofelicissimo/acc-in-sprop`

$\mathcal{T}^{=}_{\mathrm{Acc}}$ implemented on top of observational Rocq

Proof mode switch to $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ allows proofs which are easier, and much faster to check

Conclusion

We propose a design combining two theories with SProp, observational equality and Acc:

Eliminator acc-el computes up to $=$

- Has decidable type-checking
- Enjoys propositional canonicity

Eliminator acc-el computes definitionally

- Enjoys canonicity
- Conservative over $\mathcal{T}^{=}_{\mathrm{Acc}}$

Both $\mathcal{T}^{=}_{\mathrm{Acc}}$ and $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ admit set-theoretic models, and are in particular consistent

Results formalized in Rocq: `https://github.com/thiagofelicissimo/acc-in-sprop`

$\mathcal{T}^{=}_{\mathrm{Acc}}$ implemented on top of observational Rocq

Proof mode switch to $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ allows proofs which are easier, and much faster to check

Future work Apply *propositional-canonicity-by-conservativity* technique to new situations

Conclusion

We propose a design combining two theories with SProp, observational equality and Acc:

Eliminator acc-el computes up to $=$

- Has decidable type-checking
- Enjoys propositional canonicity

Eliminator acc-el computes definitionally

- Enjoys canonicity
- Conservative over $\mathcal{T}^{=}_{\mathrm{Acc}}$

Both $\mathcal{T}^{=}_{\mathrm{Acc}}$ and $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ admit set-theoretic models, and are in particular consistent

Results formalized in Rocq: `https://github.com/thiagofelicissimo/acc-in-sprop`

$\mathcal{T}^{=}_{\mathrm{Acc}}$ implemented on top of observational Rocq

Proof mode switch to $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ allows proofs which are easier, and much faster to check

Future work Apply *propositional-canonicity-by-conservativity* technique to new situations

Thank you for your attention!