Definitional Proof Irrelevance Made Accessible

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

6 March 2026 @ TallCat seminar

INRIA

Prop = Type?

Prop = Type?

Propositions

Prop = Type?

Propositions Types

Prop = Type?

Propositions ≠ Types

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

Prop = Type?

Propositions Types

Prop = Type?

Propositions = 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?

Propositions $\subset$ 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 propositions 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:

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

Ensuring compatibility of Prop with irrelevance

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

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

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:

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

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 props.:

Ensuring compatibility of Prop with irrelevance

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

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

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 props.:

1. the empty type False

Ensuring compatibility of Prop with irrelevance

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

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

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 props.:

1. the empty type False
2. the equality type =

Ensuring compatibility of Prop with irrelevance

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

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

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 props.:

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

Ensuring compatibility of Prop with irrelevance

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

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

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 props.:

1. the empty type False
2. the equality type =
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:

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

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 props.:

1. the empty type False
2. the equality type =
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

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

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

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 there is $p \equiv q$ (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

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 there is $p \equiv q$ (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

Aside from helping users, crucial for many type-theoretic constructions:

- setoid model of Altenkirch
- strict presheaves of Pédrot

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

Failure of the subsingleton criterion

Unfortunately, subsingleton criterion interacts badly with SProp...

Eliminating from SProp to Type with

- False: Still ok
- Equality (=): In presence of *function extensionality*, breaks *canonicity*, stating that

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

  Moreover, 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
- Equality (=): In presence of *function extensionality*, breaks *canonicity*, stating that

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

  Moreover, in presence of impredicativity, breaks proof normalization
- 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
- Equality (=): In presence of *function extensionality*, breaks *canonicity*, stating that

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

  Moreover, in presence of impredicativity, breaks proof normalization
- Acc: Breaks decidability of conversion/type checking (= proof checking)
  Inconsistent contexts can lie about accessibility proofs

  $$x : \mathsf{False} \vdash p : \mathsf{Acc}\ (\lambda xy.\mathsf{True})\ a$$

Failure of the subsingleton criterion

Unfortunately, subsingleton criterion interacts badly with SProp...

Eliminating from SProp to Type with

- False: Still ok
- Equality (=): In presence of *function extensionality*, breaks *canonicity*, stating that

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

  Moreover, in presence of impredicativity, breaks proof normalization
- Acc: Breaks decidability of conversion/type checking (= proof checking)
  Inconsistent contexts can lie about accessibility proofs

  $$x : \mathsf{False} \vdash p : \mathsf{Acc}\ (\lambda xy.\mathsf{True})\ a$$

  From $p : \mathsf{Acc}\ R\ a$ we can construct $\mathsf{acc\text{-}in}\ (\text{acc-inv}\ p) : \mathsf{Acc}\ R\ a$
  Definitional proof irrelevant implies they are convertible

Failure of the subsingleton criterion

Unfortunately, subsingleton criterion interacts badly with SProp...

Eliminating from SProp to Type with

- False: Still ok
- Equality (=): In presence of *function extensionality*, breaks *canonicity*, stating that

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

  Moreover, in presence of impredicativity, breaks proof normalization
- Acc: Breaks decidability of conversion/type checking (= proof checking)
  Inconsistent contexts can lie about accessibility proofs

  $$x : \mathsf{False} \vdash p : \mathsf{Acc}\ (\lambda xy.\mathsf{True})\ a$$

  From $p : \mathsf{Acc}\ R\ a$ we can construct acc-in (acc-inv p) : Acc R a
  Definitional proof irrelevant implies they are convertible
  So any acc-el can be evaluated, even if termination argument can be bogus

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

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

Undecidable type-checking until v4.19.0, implementation less stable

Now, acc-el only computes propositionally, but overall metatheory still unclear

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

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

Undecidable type-checking until v4.19.0, implementation less stable

Now, acc-el only computes propositionally, but overall metatheory still unclear

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 usual **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 usual **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 usual **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 axioms (funext, propext, etc) 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 usual **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 axioms (funext, propext, etc) cannot block reduction!

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

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

Usual **J** eliminator can be simulated using cast and transp (= **J** restricted to SProp)

This work: Reconciling Acc with SProp

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

This work: Reconciling Acc with SProp

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

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

Eliminator acc-el computes up to =

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

This work: Reconciling Acc with SProp

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

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

Eliminator acc-el computes up to =

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

But usual form of canonicity fails…

gcd 8 2 = 2 does not follow from conversion

This work: Reconciling Acc with SProp

We propose a design combining two theories that extend $\mathrm{CC}^{\mathrm{obs}}$ 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 $\mathrm{CC}^{\mathrm{obs}}$ with just constants

But usual form of canonicity fails…

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 $\mathrm{CC}^{\mathrm{obs}}$ 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 $\mathrm{CC}^{\mathrm{obs}}$ 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

gcd 8 2 = 2 follows from conversion

This work: Reconciling Acc with SProp

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

Eliminator acc-el computes up to =

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

gcd 8 2 = 2 follows from conversion

Thm $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ is conservative over $\mathcal{T}^{=}_{\mathrm{Acc}}$:
If $\vdash^{=} P : \mathrm{SProp}$ 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 $\mathrm{CC}^{\mathrm{obs}}$ with Acc:

Eliminator acc-el computes up to =

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

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

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

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

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 $\mathrm{CC}^{\mathrm{obs}}$ with Acc:

Eliminator acc-el computes up to =

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

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

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

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

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 $\mathrm{CC}^{\mathrm{obs}}$ with Acc:

Eliminator acc-el computes up to =

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

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

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

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

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!

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!

Expected to be integrated in main version of Rocq in the future

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 in 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 in Abel et al

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

Impredicativity + elimination of SProp equality into Type allows constructing 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 in Abel et al

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

Impredicativity + elimination of SProp equality into Type allows constructing 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 in Abel et al

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

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

Counter-example reproducible even on closed terms using proposition extensionality

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

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

Fundamental theorem of logical relation

Canonicity follows from *fundamental theorem*, stating that reducibility assembles into 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 follows from *fundamental theorem*, stating that reducibility assembles into 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

Fundamental theorem of logical relation

Canonicity follows from *fundamental theorem*, stating that reducibility assembles into 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

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

Fundamental theorem of logical relation

Canonicity follows from *fundamental theorem*, stating that reducibility assembles into 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

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

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

Fundamental theorem of logical relation

Canonicity follows from *fundamental theorem*, stating that reducibility assembles into 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

Example If $\Vdash t : \mathsf{Nat}$, we prove $\Vdash \mathsf{nat\text{-}rec}\ P\ p\ q\ t : P\ t$ using the fact 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 follows from *fundamental theorem*, stating that reducibility assembles into 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

Example If $\Vdash t : \mathsf{Nat}$, we prove $\Vdash \mathsf{nat\text{-}rec}\ P\ p\ q\ t : P\ t$ using the fact 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 $[\![R]\!]$

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 follows from *fundamental theorem*, stating that reducibility assembles into 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

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

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

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

Fundamental theorem of logical relation

Canonicity follows from *fundamental theorem*, stating that reducibility assembles into 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

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

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

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 $\Vdash \text{acc-el}\ P\ p\ a\ q : P\ a$ by induction on the (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!

Conservativity of $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ over $\mathcal{T}^{=}_{\mathrm{Acc}}$

We adapt Winterhalter et al's translation of *extensional type theory (ETT)* into *intensional type theory (ITT)* + funext + UIP

Conservativity of $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ over $\mathcal{T}^{=}_{\mathrm{Acc}}$

We adapt Winterhalter et al's translation of *extensional type theory (ETT)* into *intensional type theory (ITT)* + funext + UIP

Basic strategy Translate conversion in $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ to propositional equality in $\mathcal{T}^{=}_{\mathrm{Acc}}$

Application of conversion rule turned into application of cast

Conservativity of $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ over $\mathcal{T}^{=}_{\mathrm{Acc}}$

We adapt Winterhalter et al's translation of *extensional type theory (ETT)* into *intensional type theory (ITT)* + funext + UIP

Basic strategy Translate conversion in $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ to propositional equality in $\mathcal{T}^{=}_{\mathrm{Acc}}$

Application of conversion rule turned into application of cast

Problem Translated terms become *decorated* with cast

But many ways to decorate same term, translation not a function of the term

Conservativity of $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ over $\mathcal{T}^{=}_{\mathrm{Acc}}$

We adapt Winterhalter et al's translation of *extensional type theory (ETT)* into *intensional type theory (ITT)* + funext + UIP

Basic strategy Translate conversion in $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ to propositional equality in $\mathcal{T}^{=}_{\mathrm{Acc}}$

Application of conversion rule turned into application of cast

Problem Translated terms become *decorated* with cast

But many ways to decorate same term, translation not a function of the term

Key Lemma If t, u (syntactically) equal modulo cast, then they are *heterogeneously* equal in $\mathcal{T}^{=}_{\mathrm{Acc}}$

Proof uses various congruence properties of heterogeneous equality, shown with funext + UIP

Conservativity of $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ over $\mathcal{T}^{=}_{\mathrm{Acc}}$

We adapt Winterhalter et al's translation of *extensional type theory (ETT)* into *intensional type theory (ITT)* + funext + UIP

Basic strategy Translate conversion in $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ to propositional equality in $\mathcal{T}^{=}_{\mathrm{Acc}}$

Application of conversion rule turned into application of cast

Problem Translated terms become *decorated* with cast

But many ways to decorate same term, translation not a function of the term

Key Lemma If t, u (syntactically) equal modulo cast, then they are *heterogeneously* equal in $\mathcal{T}^{=}_{\mathrm{Acc}}$

Proof uses various congruence properties of heterogeneous equality, shown with funext + UIP

Theorem If $\Gamma \vdash^{\equiv} t : A$ then $\Gamma' \vdash^{=} t' : A'$, for some Γ', t', A' decorations of Γ, t, A
If $\Gamma \vdash^{\equiv} t \equiv u : A$ then $\Gamma' \vdash^{=} e : t' =_{A'} u'$, for some Γ', t', u', A' decorations of Γ, t, u, A, and some e

Conservativity of $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ over $\mathcal{T}^{=}_{\mathrm{Acc}}$

We adapt Winterhalter et al's translation of *extensional type theory (ETT)* into *intensional type theory (ITT)* + funext + UIP

Basic strategy Translate conversion in $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ to propositional equality in $\mathcal{T}^{=}_{\mathrm{Acc}}$

Application of conversion rule turned into application of cast

Problem Translated terms become *decorated* with cast

But many ways to decorate same term, translation not a function of the term

Key Lemma If t, u (syntactically) equal modulo cast, then they are *heterogeneously* equal in $\mathcal{T}^{=}_{\mathrm{Acc}}$

Proof uses various congruence properties of heterogeneous equality, shown with funext + UIP

Theorem If $\Gamma \vdash^{\equiv} t : A$ then $\Gamma' \vdash^{=} t' : A'$, for some Γ', t', A' decorations of Γ, t, A
If $\Gamma \vdash^{\equiv} t \equiv u : A$ then $\Gamma' \vdash^{=} e : t' =_{A'} u'$, for some Γ', t', u', A' decorations of Γ, t, u, A, and some e

Corollary $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ is conservative over $\mathcal{T}^{=}_{\mathrm{Acc}}$

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 have been 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 have been 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

All details can be found in preprint: `https://hal.science/hal-05474391`

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 have been 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

All details can be found in preprint: `https://hal.science/hal-05474391`

Thank you for your attention!