Definitional proof irrelevance made Accessible

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

4 May 2025

I. Context

Propositions

Agda, Lean, Rocq... are based on the Curry–Howard correspondance:

proofs and programs are really the same thing

Propositions

Agda, Lean, Rocq... are based on the Curry–Howard correspondance:

~~proofs and programs are really the same thing~~

Actually, scratch that. Sometimes it is useful to have programs and proofs as two separate things

Rocq's Propositions

Rocq has a special universe Prop for propositions

Propositions are types whose elements will be erased during extraction (e.g. equality proofs, measures of termination...)

Rocq's Propositions

Rocq has a special universe Prop for propositions

Propositions are types whose elements will be erased during extraction (e.g. equality proofs, measures of termination...)

→ Two proofs of the same proposition can never be used to produce two different results outside of Prop

Rocq's Propositions

Rocq has a special universe Prop for propositions

Propositions are types whose elements will be erased during extraction (e.g. equality proofs, measures of termination...)

→ Two proofs of the same proposition can never be used to produce two different results outside of Prop

The Subsingleton Criterion

Large elimination of an inductive proposition is only allowed when

- the proposition has at most one constructor
- all the constructor arguments are propositions

The Subsingleton Criterion

Large elimination of an inductive proposition is only allowed when

- the proposition has at most one constructor
- all the constructor arguments are propositions

These definitions fail the subsingleton criterion:

```
Inductive bool : Prop :=
| true  : bool
| false : bool
```

```
Inductive Box (A : Type) : Prop :=
| box : A → Box A
```

The Subsingleton Criterion

These definitions pass the subsingleton criterion:

The Subsingleton Criterion

These definitions pass the subsingleton criterion:

```
Inductive False : Prop :=
|
```

The Subsingleton Criterion

These definitions pass the subsingleton criterion:

```
Inductive False : Prop :=
|
```

→ extracted elimination of False = exception

The Subsingleton Criterion

These definitions pass the subsingleton criterion:

```
Inductive False : Prop :=
|
```

→ extracted elimination of False = exception

```
Inductive Eq {A : Type} (a : A) : A → Prop :=
| refl : Eq a a
```

The Subsingleton Criterion

These definitions pass the subsingleton criterion:

```
Inductive False : Prop :=
|
```

→ extracted elimination of False = exception

```
Inductive Eq {A : Type} (a : A) : A → Prop :=
| refl : Eq a a
```

→ extracted elimination of Eq = silent type casting

The Subsingleton Criterion

These definitions pass the subsingleton criterion:

```
Inductive False : Prop :=
|
```

→ extracted elimination of False = exception

```
Inductive Eq {A : Type} (a : A) : A → Prop :=
| refl : Eq a a
```

→ extracted elimination of Eq = silent type casting

```
Inductive Acc {A : Type} {R : A → A → Prop} (a : A) : A → Prop :=
| acc : (∀ (b : A). R b a → Acc b) → Acc a
```

The Subsingleton Criterion

These definitions pass the subsingleton criterion:

```
Inductive False : Prop :=
|
```

→ extracted elimination of False = exception

```
Inductive Eq {A : Type} (a : A) : A → Prop :=
| refl : Eq a a
```

→ extracted elimination of Eq = silent type casting

```
Inductive Acc {A : Type} {R : A → A → Prop} (a : A) : A → Prop :=
| acc : (∀ (b : A). R b a → Acc b) → Acc a
```

→ extracted elimination of Acc = unbounded fixpoint

Propositional Proof Irrelevance

Thus, Prop is compatible with propositional proof irrelevance:

$$PI \;:\; \forall\,(P : \mathrm{Prop})\,(x : P)\,(y : P)\,.\; x =_{\mathrm{Prop}} y$$

Propositional Proof Irrelevance

Thus, Prop is compatible with propositional proof irrelevance:

$$PI \;:\; \forall\,(P : \mathrm{Prop})\,(x : P)\,(y : P)\,.\; x =_{\mathrm{Prop}} y$$

→ after postulating *PI*, we can ignore proofs just like in HOL...

Propositional Proof Irrelevance

Thus, Prop is compatible with propositional proof irrelevance:

$$PI \;:\; \forall\,(P : \mathrm{Prop})\,(x : P)\,(y : P)\,.\; x =_{\mathrm{Prop}} y$$

→ after postulating *PI*, we can ignore proofs just like in HOL...
...up to transport hell.

Working with e.g. subset types $\Sigma\,(x : A)\,.\,P\,x$ is annoying!

Definitional Proof Irrelevance

Lean introduced definitionally proof-irrelevant propositions

SProp is also a universe of propositions, but proofs are irrelevant from within the type theory:

$$\frac{P : \mathrm{SProp} \quad x : P \quad y : P}{x \equiv y}$$

Definitional Proof Irrelevance

Surely that irrelevance business won't break anything --
after all, we can erase proofs thanks to the subsingleton criterion!

Definitional Proof Irrelevance

Surely that irrelevance business won't break anything --
after all, we can erase proofs thanks to the subsingleton criterion!

BUT there's a big difference between strict irrelevance and erasure:

Definitional Proof Irrelevance

Surely that irrelevance business won't break anything --
after all, we can erase proofs thanks to the subsingleton criterion!

BUT there's a big difference between strict irrelevance and erasure:

- after extraction, computation happens in closed terms only
  → proofs were erased from closed terms only
  → the erased proofs were telling the truth

Definitional Proof Irrelevance

Surely that irrelevance business won't break anything --
after all, we can erase proofs thanks to the subsingleton criterion!

BUT there's a big difference between strict irrelevance and erasure:

- after extraction, computation happens in closed terms only
  → proofs were erased from closed terms only
  → the erased proofs were telling the truth

- in the proof assistant, computation happens at open terms too
  → proofs are erased from potentially open terms
  → erased proofs may build upon a lie from the context!

Proof Irrelevant Subsingletons

Short review of our subsingleton criterion in light of this new info

Proof Irrelevant Subsingletons

Short review of our subsingleton criterion in light of this new info

- Large elimination of $False$ is still fine

Proof Irrelevant Subsingletons

Short review of our subsingleton criterion in light of this new info

- Large elimination of $False$ is still fine
- Large elimination of Eq means that we have to deal with type-casting between two potentially different types

Proof Irrelevant Subsingletons

Short review of our subsingleton criterion in light of this new info

- Large elimination of $False$ is still fine
- Large elimination of Eq means that we have to deal with type-casting between two potentially different types

  Lean achieves this with a modified J rule...

Proof Irrelevant Subsingletons

Short review of our subsingleton criterion in light of this new info

- Large elimination of $False$ is still fine
- Large elimination of Eq means that we have to deal with type-casting between two potentially different types

  Lean achieves this with a modified J rule...
  ...which results in a failure of normalisation

  (Abel & Coquand '19)

Proof Irrelevant Subsingletons

Short review of our subsingleton criterion in light of this new info

- Large elimination of $False$ is still fine
- Large elimination of Eq means that we have to deal with type-casting between two potentially different types

  Lean achieves this with a modified J rule...
  ...which results in a failure of normalisation

  (Abel & Coquand '19)

  Alternative strategy: switch to observational type theory, which introduces a primitive typecasting operator

  (Altenkirch, McBride, Swierstra '07 / Pujet, Tabareau '23)

Proof Irrelevant Subsingletons

Short review of our subsingleton criterion in light of this new info

Proof Irrelevant Subsingletons

Short review of our subsingleton criterion in light of this new info

- Large elimination of Acc forces us to evaluate potentially non-terminating fixpoints

Proof Irrelevant Subsingletons

Short review of our subsingleton criterion in light of this new info

- Large elimination of *Acc* forces us to evaluate potentially non-terminating fixpoints

  This time, there is no clever fix for the failure of normalisation: *Acc* renders the equational theory of Lean undecidable
  → breakage of SR, etc

  (Gilbert, Cockx, Sozeau, Tabareau '19)

Proof Irrelevant Subsingletons

Short review of our subsingleton criterion in light of this new info

- Large elimination of *Acc* forces us to evaluate potentially non-terminating fixpoints

  This time, there is no clever fix for the failure of normalisation: *Acc* renders the equational theory of Lean undecidable
  → breakage of SR, etc

  (Gilbert, Cockx, Sozeau, Tabareau '19)

  To minimise issues, Lean now discourages the use of *Acc-elim*
  → by default, recursive definitions unfold only propositionally

II. Contribution

Reconciliating SProp and Acc

Lean's solution is pragmatic, but not really satisfying

Reconciliating SProp and Acc

Lean's solution is pragmatic, but not really satisfying

- We want definitional proof irrelevance!
  Transport hell is not fun

Reconciliating SProp and Acc

Lean's solution is pragmatic, but not really satisfying

- We want definitional proof irrelevance!
  Transport hell is not fun
- We want computational accessibility!
  Without Acc, one can't even write a total evaluator for System F

Reconciliating SProp and Acc

Lean's solution is pragmatic, but not really satisfying

- We want definitional proof irrelevance!
  Transport hell is not fun
- We want computational accessibility!
  Without Acc, one can't even write a total evaluator for System F
- We want decidable typechecking!
  Leave the unpredictable heuristics ~~to the AI people~~ to the tactics

Reconciliating SProp and Acc

Lean's solution is pragmatic, but not really satisfying

- We want definitional proof irrelevance!
  Transport hell is not fun
- We want computational accessibility!
  Without Acc, one can't even write a total evaluator for System F
- We want decidable typechecking!
  Leave the unpredictable heuristics ~~to the AI people~~ to the tactics

Unfortunately, there is no way to eat our cake and have it too

Reconciliating SProp and Acc

Lean's solution is pragmatic, but not really satisfying

- We want definitional proof irrelevance!
  Transport hell is not fun
- We want computational accessibility!
  Without Acc, one can't even write a total evaluator for System F
- We want decidable typechecking!
  Leave the unpredictable heuristics ~~to the AI people~~ to the tactics

Unfortunately, there is no way to eat our cake and have it too
...unless?

Reconciliating SProp and Acc

We propose a dual approach:

We start from OTT (≈ MLTT + SProp + a typecasting operator)
and we define two extensions with an accessibility predicate

Reconciliating SProp and Acc

We propose a dual approach:

We start from OTT (≈ MLTT + SProp + a typecasting operator) and we define two extensions with an accessibility predicate

- $T^{\equiv}_{Acc}$ where the eliminator computes definitionally

  $$\text{Acc-elim}\ f\ a\ H_a \quad \Longrightarrow \quad f\ a\ (\lambda\, b\, H_b\,.\,\text{Acc-elim}\ f\ b\ H_b)$$

Reconciliating SProp and Acc

We propose a dual approach:

We start from OTT (≈ MLTT + SProp + a typecasting operator)
and we define two extensions with an accessibility predicate

- $T^{\equiv}_{Acc}$ where the eliminator computes definitionally

  $\mathrm{Acc\text{-}elim}\ f\ a\ H_a \quad \Longrightarrow \quad f\ a\ (\lambda\, b\, H_b\,.\,\mathrm{Acc\text{-}elim}\ f\ b\ H_b)$

- $T^{=}_{Acc}$ where the eliminator computes propositionally

  $\mathrm{Acc\text{-}comp} : \mathrm{Acc\text{-}elim}\ f\ a\ H_a\ =\ f\ a\ (\lambda\, b\, H_b\,.\,\mathrm{Acc\text{-}elim}\ f\ b\ H_b)$

What is the point?

Of course,

- $T^{\equiv}_{Acc}$ is undecidable as it may unfold non-terminating fixpoints
- $T^{=}_{Acc}$ loses canonicity because of the added axiom Acc-comp

However, we can prove some interesting properties

Recycling old proofs

Theorem 1

$T^{=}_{Acc}$ has decidable type-checking

Recycling old proofs

Theorem 1

$T^{=}_{Acc}$ has decidable type-checking

Proof

- $T^{=}_{Acc}$ is the same system as in Pujet, Tabareau '23
  (with a few extra constants but no additional reduction rules)

Recycling old proofs

Theorem 1

$T^{=}_{Acc}$ has decidable type-checking

Proof

- $T^{=}_{Acc}$ is the same system as in Pujet, Tabareau '23
  (with a few extra constants but no additional reduction rules)
- Therefore it is normalising, has decidable conversion and thus decidable type-checking

Canonicity with a touch of realisability

Theorem 2

$T^{\equiv}_{Acc}$ satisfies canonicity: every definition of an integer in the empty context evaluates to a proper numeral

Canonicity with a touch of realisability

Theorem 2

$T^{\equiv}_{Acc}$ satisfies canonicity: every definition of an integer in the empty context evaluates to a proper numeral

Proof

- $T^{\equiv}_{Acc}$ is sound (we can construct a model in IZF set theory)

Canonicity with a touch of realisability

Theorem 2

$T^{\equiv}_{Acc}$ satisfies canonicity: every definition of an integer in the empty context evaluates to a proper numeral

Proof

- $T^{\equiv}_{Acc}$ is sound (we can construct a model in IZF set theory)
- consequence: if $T^{\equiv}_{Acc} \vdash H_a : Acc\ a$, then the call graph of Acc-$elim\ f\ a\ H_a$ is well founded

Canonicity with a touch of realisability

Theorem 2

$T^{\equiv}_{Acc}$ satisfies canonicity: every definition of an integer in the empty context evaluates to a proper numeral

Proof

- $T^{\equiv}_{Acc}$ is sound (we can construct a model in IZF set theory)
- consequence: if $T^{\equiv}_{Acc} \vdash H_a : Acc\ a$, then the call graph of Acc-*elim* $f\ a\ H_a$ is well founded
- from there, we can follow a standard logical relations argument to derive canonicity

Transport hell works, at least in theory

Theorem 3

$T^{\equiv}_{Acc}$ is conservative over $T^{=}_{Acc}$: if A is a type of $T^{=}_{Acc}$ which is provable in $T^{\equiv}_{Acc}$, then it is also provable in $T^{=}_{Acc}$.

Transport hell works, at least in theory

Theorem 3

$T^{\equiv}_{Acc}$ is conservative over $T^{=}_{Acc}$: if A is a type of $T^{=}_{Acc}$ which is provable in $T^{\equiv}_{Acc}$, then it is also provable in $T^{=}_{Acc}$.

Proof

- Both of our theories are extensions of observational type theory, meaning that we have funext and UIP

Transport hell works, at least in theory

Theorem 3

$T^{\equiv}_{Acc}$ is conservative over $T^{=}_{Acc}$: if A is a type of $T^{=}_{Acc}$ which is provable in $T^{\equiv}_{Acc}$, then it is also provable in $T^{=}_{Acc}$.

Proof

- Both of our theories are extensions of observational type theory, meaning that we have funext and UIP
- $T^{\equiv}_{Acc}$ only has one additional definitional equality. Thus we can follow the argument for conservativity of ETT over ITT from Winterhalter, Sozeau, Tabareau '20

Summary

Thus we have two theories which prove the same statements

- $T^{\equiv}_{Acc}$ has canonicity but is undecidable
- $T^{=}_{Acc}$ is decidable but has only homotopy canonicity

A unified implementation

We implemented $T^{=}_{Acc}$ in Rocq using its sort SProp

A unified implementation

We implemented $T^{=}_{Acc}$ in Rocq using its sort SProp

- When the user wants to prove a proposition, they can locally turn on definitional Acc unfolding to get a shorter proof

A unified implementation

We implemented $T^{=}_{Acc}$ in Rocq using its sort SProp

- When the user wants to prove a proposition, they can locally turn on definitional Acc unfolding to get a shorter proof
- At the end of their proof, they know that the proof is provable in the propositional system by conservativity

A unified implementation

We implemented $T^{=}_{Acc}$ in Rocq using its sort SProp

- When the user wants to prove a proposition, they can locally turn on definitional Acc unfolding to get a shorter proof
- At the end of their proof, they know that the proof is provable in the propositional system by conservativity
- Since propositions are proof-irrelevant, we might as well add the proved prop as an axiom – it will not interfere with computations!

TL;DR

We have turned definitional Acc unfolding into a tactic (spiritually)

Thank you!