Definitional Proof-Irrelevance without K

Matthieu Sozeau, $\pi.r^2$, Inria Paris & IRIF
joint work with Jesper Cockx (Chalmers, Sweden),
Gaëtan Gilbert and Nicolas Tabareau (Gallinette, Inria Nantes)

Séminaire Deducteam
January 31st 2019
ENS Cachan

Definitional Proof-Irrelevance without K

1. Propositions and equality
   - Propositions in Homotopy Type Theory
   - Propositions in the Calculus of Inductive Constructions
   - Extraction and singleton elimination

2. A universe of strict propositions
   - The `SProp` universe
   - Semantics & Implementation

Notions of Equality

Type Theory has two notions of equality:

- Definitional equality: $x \equiv y : A$ for $x, y : A$. Equality of computations: includes β reduction, reduction of fixpoints and pattern-matching, rewrite rules etc...
  - Proof-irrelevant, strict notion: no witnesses. "Equality on the nose"
  - Not a type: we cannot "assume" definitional equalities
- (Propositional/Path) equality: $x =_A y$ for $x, y : A$.
  - A type with a single constructor $\mathsf{id}_a : a =_A a$.
  - Coincides with definitional equality in the empty context only.
  - Can be inhabited by arbitrary terms.

Example:
$n * m =_{\mathbb{N}} m * n$ is provable by induction, but in general
$n * m \not\equiv m * n$.

Equality in mathematics

Standard mathematical reasoning and the usual notion of equality of data structures in computer science is rather a strict notion:

$$\begin{array}{lcl} a, b & \in & \mathbb{N} \\ ? & \in & a = b \end{array}$$

Equality in mathematics

Standard mathematical reasoning and the usual notion of equality of data structures in computer science is rather a strict notion:

$$\begin{array}{lcll} a, b & \in & \mathbb{N} & \\ a = b & \leftrightarrow & \forall P, P\ a \to P\ b & \text{(Leibniz principle)} \end{array}$$

Equality is a property (as opposed to a structure)
The same holds for the order relations on natural numbers $n \leq m$.

Extensional Type Theory

One way to recover the strict notion of equality in type theory:

$$\frac{\Gamma \vdash p : T = U : A}{\Gamma \vdash T \equiv U : A}\ \textsc{Reflection}$$

Extensional Type Theory

One way to recover the strict notion of equality in type theory:

$$\frac{\Gamma \vdash p : T = U : A}{\Gamma \vdash T \equiv U : A}\ \textsc{Reflection}$$

- Breaks decidability of type checking
- Introduces uniqueness of identity proofs:

$$\forall (x\ y : A)(p\ q : x = y), p = q \quad \text{(UIP)}$$

Strict Propositions

A strict proposition P (not necessarily equality) shall have the property:

$$\forall (x\ y : P), x \equiv y \quad \text{(Definitional irrelevance)}$$

By definition of the identity type, it will also enjoy

$$\forall (x\ y : P), x =_P y \quad \text{(Propositional irrelevance)}$$

But we don't necessarily want UIP!

The Setoid Model: justifying functional extensionality

"Our solution is also based on the setoid model but as the metatheory, where the construction takes place, we use an extension of Intensional Type Theory by a universe of propositions, such that all proofs of a proposition are definitionally equal."

Hofmann'95 & Altenkirch'99

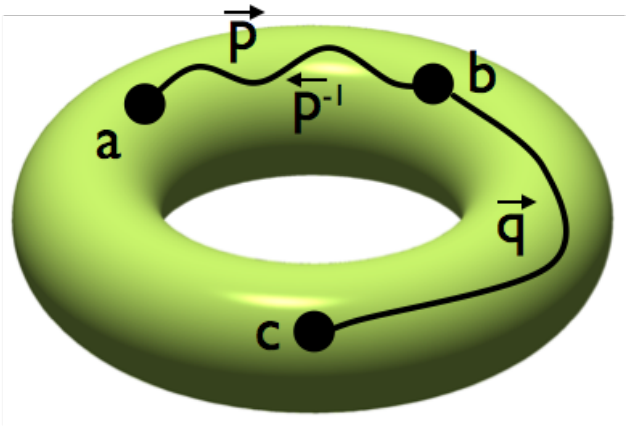
Definitional Proof-Irrelevance without K

1. Propositions and equality
   - Propositions in Homotopy Type Theory
   - Propositions in the Calculus of Inductive Constructions
   - Extraction and singleton elimination
2. A universe of strict propositions
   - The `SProp` universe
   - Semantics & Implementation

Homotopy Type Theory and Proof Relevance

Key idea of Homotopy Type Theory:

identity type $\leftrightarrow$ ∞-groupoid structure



a, b, c	:	$\mathbb{T}$
p	:	$a = b$
id_a	:	$a = a$
$\_^{-1}$	:	$a = b \to b = a$
$invsym$	:	$p^{-1} \circ p =_{a=a} id_a$

$+$ an infinity of laws

The identity type naturally represents a weak structure.

Homotopy Type Theory and Proof Relevance

Voevodsky introduced an internalization of the notion of proposition using the homotopy level h-level(n) of a type:

$$\begin{aligned} \bot_{hprop} &: \mathsf{h\text{-}level}(1)(\bot) \\ &: \mathsf{hProp}(\bot) \\ &\equiv \forall(x\ y : \bot), x = y \end{aligned}$$

- A homotopy (mere) proposition P is propositionally irrelevant.

Homotopy Type Theory and Proof Relevance

Voevodsky introduced an internalization of the notion of proposition using the homotopy level h-level(n) of a type:

$$\begin{aligned} \bot_{hprop} &: \mathsf{h\text{-}level}(1)(\bot) \\ &: \mathsf{hProp}(\bot) \\ &\equiv \forall(x\ y : \bot), x = y \end{aligned}$$

- A homotopy (mere) proposition P is propositionally irrelevant.

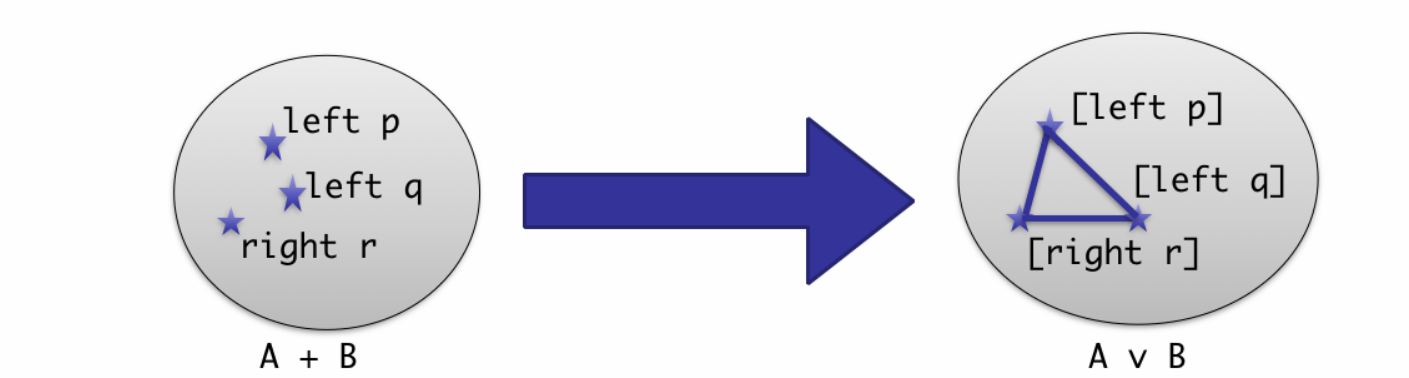
$$\begin{aligned} \mathbb{N}_{hset} &: \mathsf{h\text{-}level}(2)(\mathbb{N}) \\ &\equiv \mathsf{hSet}(\mathbb{N}) \\ &\equiv \forall(x\ y : \mathbb{N}), \mathsf{hProp}(x = y) \\ &\equiv \forall(x\ y : \mathbb{N})(p\ q : x = y), p =_{x=y} q \end{aligned}$$

- hSets correspond to ordinary sets, but with a weak equality.
- h-level$(n) \leftrightarrow (n-2)$-truncated type

Truncation

(-1)-Truncation turns any type into an hProp, similarly to the bracket types of Awodey and Bauer:

$$\textsc{Trunc}\ \frac{A : \mathtt{Type} \quad x : A}{[x] : \mathsf{Trunc}\ A} \qquad \textsc{Trunc-Eq}\ \frac{A : \mathtt{Type} \quad x, y : \mathsf{Trunc}\ A}{\mathsf{trunc\text{-}eq}\ x\ y : x = y}$$



where $A \vee B = \mathsf{Trunc}\ (A + B)$

Truncation elimination

Its elimination principle is restricted to other hProps:

$$\textsc{Trunc-elim}\quad \frac{A : \texttt{Type} \quad t : \mathsf{Trunc}\ A \qquad P : \mathsf{Trunc}\ A \to \mathsf{hProp} \quad e : \forall (a : A), P\ [a]}{\mathsf{trunc\text{-}elim}\ A\ P\ x\ e : P\ x}$$

Idea: Once truncated, we can "look into" the content of $[x]$ only to build other propositions, which are by definition proof-irrelevant.

hProp

In HoTT, hProp contains:

- Any contractible ((-2)-truncated) type: $\top$, singleton types Sing $A\ a := \Sigma x : A, x = a$
- $\bot$
- Dependent pairs of hProps
- (-1)-truncated types
- Any type that can be shown equivalent to an hProp.

hProp

In HoTT, hProp contains:

- Any contractible ((-2)-truncated) type: $\top$, singleton types Sing $A\ a := \Sigma x : A, x = a$
- $\bot$
- Dependent pairs of hProps
- (-1)-truncated types
- Any type that can be shown equivalent to an hProp.

Way too large for a strict interpretation: e.g:

$$p\ q : \text{Sing}\ A\ a \qquad\qquad \vdash \quad p \equiv q$$

hProp

In HoTT, hProp contains:

- Any contractible ((-2)-truncated) type: $\top$, singleton types Sing $A\ a := \Sigma x : A, x = a$
- $\bot$
- Dependent pairs of hProps
- (-1)-truncated types
- Any type that can be shown equivalent to an hProp.

Way too large for a strict interpretation: e.g:

$$\begin{array}{lll} & p\ q : \text{Sing}\ A\ a & \vdash \quad p \equiv q \\ \Rightarrow & x\ y : A, p : x = a, q : y = a & \vdash \quad (x, p) \equiv (y, q) \end{array}$$

hProp

In HoTT, hProp contains:

- Any contractible ((-2)-truncated) type: $\top$, singleton types Sing $A\ a := \Sigma x : A, x = a$
- $\bot$
- Dependent pairs of hProps
- (-1)-truncated types
- Any type that can be shown equivalent to an hProp.

Way too large for a strict interpretation: e.g:

$$\begin{array}{lll} & p\ q : \text{Sing}\ A\ a & \vdash \quad p \equiv q \\ \Rightarrow & x\ y : A, p : x = a, q : y = a & \vdash \quad (x, p) \equiv (y, q) \\ \Rightarrow & x\ y : A, p : x = a, q : y = a & \vdash \quad (x, p).1 \equiv (y, q).1 \end{array}$$

hProp

In HoTT, hProp contains:

- Any contractible ((-2)-truncated) type: $\top$, singleton types Sing $A\ a := \Sigma x : A, x = a$
- $\bot$
- Dependent pairs of hProps
- (-1)-truncated types
- Any type that can be shown equivalent to an hProp.

Way too large for a strict interpretation: e.g:

$$
\begin{array}{lll}
 & p\ q : \text{Sing}\ A\ a & \vdash \quad p \equiv q \\
\Rightarrow & x\ y : A, p : x = a, q : y = a & \vdash \quad (x, p) \equiv (y, q) \\
\Rightarrow & x\ y : A, p : x = a, q : y = a & \vdash \quad (x, p).1 \equiv (y, q).1 \\
\Rightarrow & x\ y : A, p : x = a, q : y = a & \vdash \quad x \equiv y
\end{array}
$$

Definitional Proof-Irrelevance without K

1. Propositions and equality
   - Propositions in Homotopy Type Theory
   - Propositions in the Calculus of Inductive Constructions
   - Extraction and singleton elimination

2. A universe of strict propositions
   - The `SProp` universe
   - Semantics & Implementation

A detour to Coq: The Prop sort

In the Calculus of Inductive Constructions, Prop is used to represent propositions.

```
Inductive ≤ : ℕ → ℕ → Prop :=
  ≤0 : ∀ n, 0 ≤ n
| ≤S : ∀ m n, m ≤ n → S m ≤ S n.
```

A detour to Coq: The Prop sort

In the Calculus of Inductive Constructions, Prop is used to represent propositions.

```
Inductive ≤ : ℕ → ℕ → Prop :=
  ≤0 : ∀ n, 0 ≤ n
| ≤S : ∀ m n, m ≤ n → S m ≤ S n.

Definition boundedℕ (k : ℕ) : Type := { n : ℕ & n ≤ k }.

Definition add {k} (n m : boundedℕ k) (e : n + m ≤ k) : boundedℕ k := ( n + m ; e).
```

Propositional proof irrelevance

Only propositionally proof-irrelevant:

```
Definition bounded_add_associativity k (n m p: boundedℕ k) e1 e2 e'1 e'2 :
  add (add n m e1) p e2 = add n (add m p e'1) e'2.
```

```
k : ℕ
n, m, p : boundedℕ k
e2, e2' : n + m + p <= k
============================
(n + m + p; e2) = (n + m + p; e2')
```

Propositional proof irrelevance

Only propositionally proof-irrelevant:

Definition bounded_add_associativity k (n m p: bounded$\mathbb{N}$ k) e_1 e_2 e'_1 e'_2 :
add (add n m e_1) p e_2 = add n (add m p e'_1) e'_2.

```
k : ℕ
n, m, p : boundedℕ k
e2, e2' : n + m + p <= k
============================
(n + m + p; e2) = (n + m + p; e2')
```

In general:

- $p, q : x \leq y$ not equal propositionally
- Even so, $(x, p) =_{\{x:\mathbb{N} \mid x \leq y\}} (x, q)$ is not easy to work with (transports and "setoid hell")

Prop and hProp

Luckily here we can show:

```
Equations ≤_hprop {m n} (e e' : m ≤ n) : e = e' :=
  ≤_hprop (≤0 _) (≤0 _) := eq_refl;
  ≤_hprop (≤S _ _ e) (≤S n m e') := ap (≤S n m) (≤_hprop e e').
```

- But inhabitants of Prop are not hProps in general, e.g. disjunctions.
- Have to resort to an axiom of proof-irrelevance, destroying canonicity.

Prop and Extraction

Prop is used to model proof terms that are erasable by extraction.

Idea: extraction $\mathcal{E}(t)$ of $\vdash t : \mathbb{N}$, removes all the propositional content from t, s.t.:

$$\text{If } t \rightsquigarrow^{whnf} \underline{n} \text{ then } \mathcal{E}(t) \rightsquigarrow^{whnf} \underline{n}.$$

Assumption: t is closed.

Example:

- removes proofs $n \leq k$ above
- addition of bounded naturals $\rightarrow$ addition of naturals

Singleton elimination

- `Prop` is secluded from `Type` to allow extraction.
- But not completely separate!

One can eliminate an inductive object from `Prop` (e.g. a proof of $n \leq m$) to `Type` iff:

- The inductive has at most one constructor.
- All its arguments are propositions.

Informally: closed terms of these types carry no computational content.

Singleton elimination II

Singleton elimination allows to eliminate:

- True, False, and (dependent) pairs of propositions
- eq: the equality type in `Prop`
- Acc: accessibility proofs

Also too large for a strict interpretation!

Singleton elimination disallows to eliminate:

- Trunc defined in `Prop`, as its constructor has an argument in `Type`, as expected.
- The definition of le on natural numbers, because it has two constructors (le0, leS). However it is an hProp...

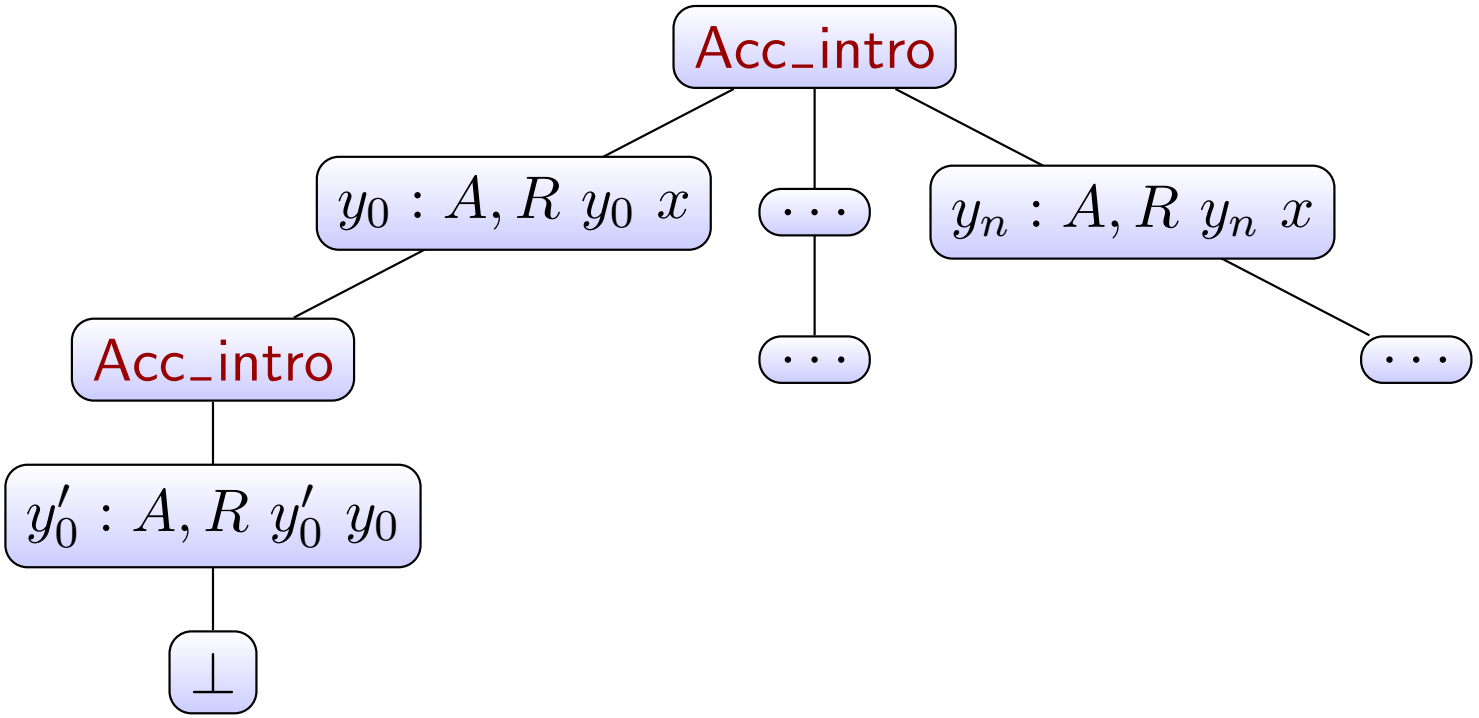
Singleton elimination of equality

- Assumes that equality carries no computational content (strict interpretation).
- (Weak) equalities can contain isomorphisms/equivalences when we assume univalence.

$\Rightarrow$ Incompatible with HoTT.

Singleton elimination of accessibility

```
Inductive Acc (A : Type) (R : A → A → Prop) (x : A) : Prop :=
  Acc_intro : (∀ y : A, R y x → Acc R y) → Acc R x
```



Definitional irrelevance of Acc $\Rightarrow$ undecidable type-checking (bug in Lean).

$\Rightarrow$ Not a strict proposition

The case of less-than or equal

le is an hProp but has two constructors $\Rightarrow$ singleton elimination disallows its elimination to `Type`.

$$
\begin{aligned}
&\texttt{Inductive} \leq : \mathbb{N} \to \mathbb{N} \to \texttt{Prop} := \\
&\quad \leq 0 : \forall\ \mathsf{n},\ 0 \leq \mathsf{n} \\
&\mid \leq \texttt{S} : \forall\ \mathtt{m}\ \mathsf{n},\ \mathtt{m} \leq \mathsf{n} \to \texttt{S}\ \mathtt{m} \leq \texttt{S}\ \mathsf{n}.
\end{aligned}
$$

However, it is propositional:

- constructors are orthgonal
- the induction is structurally justified
- only one, canonical proof of $\underline{n} \leq \underline{m}$ determined by $\underline{n}, \underline{m}$.

Dependent Pattern-Matching to the Rescue

Idea: Compile by dependent-pattern matching on the indices

Syntactic criteria for "invertibility":

1. Every non-forced argument of a constructor must be in `SProp`: no computational content can escape from `SProp`.

$$
\begin{aligned}
&\texttt{Inductive} \le : \mathbb{N} \to \mathbb{N} \to \texttt{Prop} := \\
&\quad \le 0 : \forall\ \mathtt{n},\ 0 \le \mathtt{n} \\
&\mid \le \texttt{S} : \forall\ \mathtt{m}\ \mathtt{n},\ \mathtt{m} \le \mathtt{n} \to \texttt{S}\ \mathtt{m} \le \texttt{S}\ \mathtt{n}.
\end{aligned}
$$

Dependent Pattern-Matching to the Rescue

Idea: Compile by dependent-pattern matching on the indices

Syntactic criteria for "invertibility":

1. Every non-forced argument of a constructor must be in `SProp`: no computational content can escape from `SProp`.
2. The return types of constructors must be pairwise orthogonal: indices uniquely determine constructors.

$$
\begin{aligned}
&\texttt{Inductive} \le : \mathbb{N} \to \mathbb{N} \to \texttt{Prop} := \\
&\quad \le 0 : \forall\, n, 0 \le n \\
&\mid \le \texttt{S} : \forall\, m\ n, m \le n \to \texttt{S}\ m \le \texttt{S}\ n.
\end{aligned}
$$

Dependent Pattern-Matching to the Rescue

Idea: Compile by dependent-pattern matching on the indices

Syntactic criteria for "invertibility":

1. Every non-forced argument of a constructor must be in `SProp`: no computational content can escape from `SProp`.
2. The return types of constructors must be pairwise orthogonal: indices uniquely determine constructors.
3. Every inductive reference should be syntactically guarded: rules out accessibility.

$$
\begin{aligned}
&\texttt{Inductive} \le : \mathbb{N} \to \mathbb{N} \to \texttt{Prop} := \\
&\quad \le 0 : \forall\, n,\, 0 \le n \\
&\mid \le \texttt{S} : \forall\, m\ n,\, m \le n \to \texttt{S}\ m \le \texttt{S}\ n.
\end{aligned}
$$

Dependent Pattern-Matching to the Rescue

Idea: Compile by dependent-pattern matching on the indices

Syntactic criteria for "invertibility":

1. Every non-forced argument of a constructor must be in `SProp`: no computational content can escape from `SProp`.
2. The return types of constructors must be pairwise orthogonal: indices uniquely determine constructors.
3. Every inductive reference should be syntactically guarded: rules out accessibility.

$$\begin{aligned}&\texttt{Inductive} \le : \mathbb{N} \to \mathbb{N} \to \texttt{Prop} :=\\ &\quad \le 0 : \forall\ \texttt{n}, 0 \le \texttt{n}\\ &\mid \le \texttt{S} : \forall\ \texttt{m}\ \texttt{n}, \texttt{m} \le \texttt{n} \to \texttt{S}\ \texttt{m} \le \texttt{S}\ \texttt{n}.\end{aligned}$$

$$m \le n := \texttt{case}_m \left\{ \begin{array}{lcl} 0 & \mapsto & \le 0 \\ \texttt{S}\ m' & \mapsto & \texttt{case}_n \left\{ \begin{array}{lcl} 0 & \mapsto & \bot \\ \texttt{S}\ n' & \mapsto & \le \texttt{S}\ (p : m' \le n') \end{array} \right. \end{array} \right.$$

The bad le

```
Inductive ≤bad : ℕ → ℕ → SProp :=
| ≤bad_refl: ∀ n, n ≤bad n
| ≤badS: ∀ m n, m ≤bad n → m ≤bad S n.
```

In the absurd context $S\ n \leq$bad n we have two proofs of $S\ n \leq$bad $S\ n$:

- $\leq$`bad_refl` $(S\ n)$
- $\leq$`badS` $(S\ n)\ n\ e$

Cannot eta-expand a case analysis on $e : S\ n \leq$bad $S\ n$ to a canonical constructor, so not a natural `SProp`.

Definitional Proof-Irrelevance without K

1. Propositions and equality

2. A universe of strict propositions
   - The `SProp` universe
   - Semantics & Implementation

The SProp universe

$$\frac{}{\mathtt{SProp}_i : \mathtt{Type}_{i+1}}$$

Predicative (in Agda) or impredicative (in Coq).

- SProp enjoys definitional proof-irrelevance:

$$\frac{A : \mathtt{SProp} \quad x, y : A}{x \equiv y : A}$$

- The extended theory is independent from UIP or Univalence.
- Can encode all types in Prop except for accessibility and equality.

Main ideas of SProp

SProp: a syntactic approximation of mere propositions (hProps).

- Closed by dependent products whose codomain is in SProp
- sEmpty and (dependent) pairs of SProps.
- A truncation operation Squash that turns any type into an SProp.
- Every natural (non-truncated) SProp can be eliminated to Type.

That's it! Inductive types are treated by the transformation into fixpoints.

The Strict Empty type

$$\frac{}{\mathsf{sEmpty} : \mathtt{SProp}} \qquad \frac{A : \mathtt{Type} \quad e : \mathsf{sEmpty}}{\mathrm{sEmpty\text{-}elim}\ e : A}$$

E.g. the strict unit type can be defined as:

$$\mathsf{sUnit} : \mathtt{SProp} := \mathsf{False} \to \mathsf{False}$$

Squash

The Squash operation mimicks the truncation operator of HoTT:

$$\textsc{SquashForm}\ \frac{A : \texttt{Type}}{\mathsf{Squash}\ A : \texttt{SProp}} \qquad \textsc{Squash}\ \frac{A : \texttt{Type} \quad x : A}{\mathsf{sq}\ x : \mathsf{Squash}\ A}$$

$$\textsc{Unsquash}\ \frac{A : \texttt{Type} \quad t : \mathsf{Squash}\ A \qquad P : \texttt{SProp} \quad f : A \to P}{\mathsf{unsquash}\ A\ P\ t\ f : P}$$

Thanks to irrelevance, dependent elimination can be derived from the non-dependent one.

Box

An inverse Box operator can be used to inject SProp into Type (no $\texttt{SProp} \leq \texttt{Type}$ cumulativity)

$$\textsc{Box}\ \frac{A : \texttt{SProp}_i}{\Box A : \texttt{Type}_i} \qquad \textsc{BoxIntro}\ \frac{A : \texttt{SProp} \quad x : A}{\mathsf{box}\ x : \Box A}$$

$$\textsc{BoxElim}\ \frac{A : \texttt{SProp} \quad P : A \to \texttt{SProp}/\texttt{Type} \quad x : A \vdash f : P\ (\mathsf{box}\ x) \quad b : \Box A}{\mathsf{unbox}\ A\ P\ f\ b : P\ b}$$

Conversion: $\mathsf{unbox}\ A\ P\ f\ (\mathsf{box}\ b) = f\ b$.
Actually encoded using inductive types which are allowed to carry sprops and still live in type.

Inductive types in SProp

- Only allow inductive types whose arguments are all in SProp. Includes sigma-types.
- Otherwise, encode them using fixpoints, False, dependent pairs and Squash.

$$\leq : \mathbb{N} \to \mathbb{N} \to \texttt{SProp}$$

$$(x, p) \equiv (x, q) : \{x : \mathbb{N} \mid x \leq y\} \text{ for all } p, q$$

Implementation

- In Coq: using irrelevance marks on binders for untyped conversion.
- In Agda: using type-based conversion and a predicative hierarchy
  $\Rightarrow$ fixes the unsound irrelevant arguments system

Both to be integrated in the next versions of the proof assistants!

Metatheory

- Decidability of type-checking: extension of Abel & Scherer, 2012. This requires using marks in the terms for `SProp` content.
- Consistency: by a syntactic model in Extensional Type Theory:

$$[\![A : \texttt{SProp}]\!] = \Sigma\ A : \texttt{Type}.\forall x\ y : A, x = y$$

  $\Rightarrow$ Safe and modular extension with UIP
- Independence from UIP: by a syntactic model in HTS/2-level type theory.
  $\Rightarrow$ Compatible with univalence

Applications

- A simpler definition of primality in math-comp:

  ```
  Inductive prime (p:ℕ) : SProp :=
   prime_intro : 1 < p → (∀ n, 1 ≤ n → n < p → is_gcd n p 1) → prime p.
  ```

  Compare with Eratosthene's sieve implementation + correctness proof.
- Initially Conjecture
  https://github.com/guillaumebrunerie/initiality/
- Setoid Type Theory
  https://bitbucket.org/akaposi/setoid
- Forcing with Cubical Types
  https://github.com/loic-p/cubical_forcing/

Conclusion

Our new universe of strict propositions:

- Fixes issues with `Prop` and singleton eliminations to allow definitional irrelevance without introducing axioms.
- Eases programming with subset types, decidable propositions (all included in `SProp`) and proof-irrelevant types in general.
- Is HoTT-compatible: `SProp` models the equality of Bishop sets (as shown by Coquand), a fibrant universe in the cubical sets model of Homotopy Type Theory.

Future work

Open problem: internalization of simplicial types and higher categorical structures inside homotopy type theory.

Existing solutions require an extension with a strict equality:

- HTS (Voevodsky): add a new, strict equality type using reflection.
- 2-level type theory (Anenkov, Caprioti and Kraus): consider a homotopy type theory inside a theory with uniqueness of identity proofs.
- Lambdapi: reflection restricted to the decidable theory of natural numbers?

Question: is `SProp` "just enough"?
Partial answer: naïvely adding a strict equality introduces UIP at every type. We rather need a notion of strict sets.