A Universe of Strict Propositions in Type Theory

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

Journées PPS
November 9th 2018
IRIF, Paris
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 $\text{SProp}$ 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 $\beta$ reduction, reduction of fixpoints and pattern-matching 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 $\text{id}_a : a =_A a$.
  - 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:

\[ a, b \in \mathbb{N} \quad ? \in a = b \]
Standard mathematical reasoning and the usual notion of equality of data structures in computer science is rather a strict notion:

\[ a, b \in \mathbb{N} \]
\[ a = b \iff \forall P, P \ a \rightarrow P \ b \] (Leibniz principle)

Equality is a property (as opposed to a structure)
One way to recover the strict notion of equality in type theory:

\[
\begin{align*}
\text{Reflection} \\
\Gamma \vdash p : T = U : A \\
\Gamma \vdash T \equiv U : A
\end{align*}
\]
One way to recover the strict notion of equality in type theory:

\[
\text{Reflection}
\begin{align*}
\Gamma & \vdash p : T = U : A \\
\Gamma & \vdash T \equiv U : A
\end{align*}
\]

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

\[
\forall (x \ y : A)(p \ q : x = y), p = q \quad \text{(UIP)}
\]
A strict proposition $P$ (not necessarily equality) shall have the property:

$$\forall(x \ y : P), x \equiv y$$  \hspace{1cm} \text{(Definitional irrelevance)}

By definition of the identity type, it will also enjoy

$$\forall(x \ y : P), x =_P y$$ \hspace{1cm} \text{(Propositional irrelevance)}

But we don’t necessarily want UIP!
A Universe of Strict Propositions in Type Theory

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 $\text{SProp}$ universe
   - Semantics & Implementation
Key idea of Homotopy Type Theory:

identity type $\leftrightarrow \infty$-groupoid structure

+a, b, c : $\mathbb{T}$
+p : $a = b$
+id$_a$ : $a = a$
+_1 : $a = b \rightarrow b = a$
+invsym : $p^{-1} \circ p =_{a=a} id_a$
+
+an infinity of laws

The identity type naturally represents a weak structure.
Voevodsky introduced an internalization of the notion of proposition using the homotopy level $\text{h-level}(n)$ of a type:

$$\bot_{\text{hprop}} : \text{h-level}(1)(\bot)$$

$$\text{hProp}(\bot) \equiv \forall (x y : \bot), x = y$$

A homotopy (mere) proposition $P$ is propositionally irrelevant.
Voevodsky introduced an internalization of the notion of proposition using the homotopy level $h$-level$(n)$ of a type:

$$
\bot_{hprop} : h\text{-level}(1)(\bot) \\
: h\text{Prop}(\bot) \\
\equiv \forall (x \ y : \bot), x = y
$$

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

$$
\mathbb{N}_{hset} : h\text{-level}(2)(\mathbb{N}) \\
\equiv h\text{Set}(\mathbb{N}) \\
\equiv \forall (x \ y : \mathbb{N}), h\text{Prop}(x = y) \\
\equiv \forall (x \ y : \mathbb{N})(p \ q : x = y), p =_{x=y} q
$$

- hSets correspond to ordinary sets, but with a weak equality.
- $h$-level$(n) \leftrightarrow (n - 2)$-truncated type
(−1)-Truncation turns any type into an hProp, similarly to the bracket types of Awodey and Bauer:

\[
\begin{align*}
\text{Trunc} & : A : \text{Type} \quad x : A \\
\quad & \quad \quad \quad \quad \quad \quad \rightarrow [x] : \text{Trunc } A \\
\text{Trunc-Eq} & : A : \text{Type} \quad x, y : \text{Trunc } A \\
\quad & \quad \quad \quad \quad \quad \rightarrow \text{trunc-eq } x y : x = y
\end{align*}
\]

where \( A \lor B = \text{Trunc } (A + B) \)
Truncation elimination

Its elimination principle is restricted to other hProps:

\[
\text{TRUNC-ELIM} \\
A : \text{Type} \quad t : \text{Trunc } A \\
P : \text{Trunc } A \rightarrow \text{hProp} \quad e : \forall (a : A), P \, [a] \\
\text{trunc-elim } A \, P \, x \, e : P \, x
\]

Idea: Once truncated, we can “look into” the content of \([x]\) only to build other propositions, which are by definition proof-irrelevant.
In HoTT, hProp contains:

- Any contractible ((−2)-truncated) type: \( \top \), singleton types
  \( \text{Sing } A \ a := \Sigma x : A, x = a \)

- \( \bot \)

- Dependent pairs of hProps

- ((−1)-truncated types)

- Any type that can be shown equivalent to an hProp.
In HoTT, hProp contains:

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

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

\[
\begin{align*}
p \ q & : \text{Sing } A \ a \\ \Rightarrow & \ x \ y : A, p : x = a, q : y = a \vdash p \equiv q \\
\Rightarrow & \ x \ y : A, p : x = a, q : y = a \vdash (x, p) \equiv (y, q) \\
\Rightarrow & \ x \ y : A, p : x = a, q : y = a \vdash (x, p).1 \equiv (y, q).1 \\
\Rightarrow & \ x \ y : A, p : x = a, q : y = a \vdash x \equiv y
\end{align*}
\]
A Universe of Strict Propositions in Type Theory

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 \textit{SProp} universe
   - Semantics & Implementation
In the calculus of inductive constructions, Prop is used to represent propositions.

\[
\text{Inductive } \leq : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \text{Prop} := \\
\leq 0 : \forall n, 0 \leq n \\
| \leq S : \forall m n, m \leq n \rightarrow S m \leq S n.
\]
A detour to CoQ: The \( \text{Prop} \) sort

In the calculus of inductive constructions, \( \text{Prop} \) is used to represent propositions.

\[
\text{Inductive} \quad \leq : \mathbb{N} \to \mathbb{N} \to \text{Prop} :=
\begin{align*}
\leq 0 : & \forall n, 0 \leq n \\
| \leq S : & \forall m n, m \leq n \rightarrow S m \leq S n.
\end{align*}
\]

\textbf{Definition} \quad \text{bounded}\(\mathbb{N}\) \((k : \mathbb{N}) : \text{Type} := \{n : \mathbb{N} \& n \leq k\}.

\textbf{Definition} \quad \text{add \{k\}} \(n m : \text{bounded}\(\mathbb{N}\) k) (e : n + m \leq k) : \text{bounded}\(\mathbb{N}\) k := (n + m ; e).
Only propositionally proof-irrelevant:

Definition bounded_add_assocativity k (n m p: boundedℕ k) e₁ e₂ e₁' e₂' :
  add (add n m e₁) p e₂ = add n (add m p e₁') e₂'.

In general:

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

Luckily here we can show:

Equations \( \leq_{hprop} \{m \ n\} \ (e \ e' : m \leq n) : e = e' := \)
\( \leq_{hprop} (\leq 0 \_ ) (\leq 0 \_ ) := eq\_refl ; \)
\( \leq_{hprop} (\leq S \_ \_ e) (\leq S n m e') := ap (\leq S n m) (\leq_{hprop} e e'). \)

- But inhabitants of Prop are not hProps in general.
- Have to resort to an axiom of proof-irrelevance, destroying canonicity.
Prop is used to model proof terms that are erasable by extraction.

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

$$\text{If } t \rightsquigarrow_{\text{whnf}} n \text{ then } E(t) \rightsquigarrow_{\text{whnf}} n.$$

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 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.

⇒ Incompatible with HoTT.
Inductive Acc (A : Type) (R : A → A → Prop) (x : A) : Prop :=
Acc_intro : (∀ y : A, R y x → Acc R y) → Acc R x

Singleton elimination of accessibility

Definitional irrelevance of Acc ⇒ undecidable type-checking (bug in Lean).
⇒ Not a strict propositions
The case of less-than or equal

```latex
\text{le} \text{ is an hProp but has two constructors } \Rightarrow \text{ singleton elimination disallows its elimination to } \text{Type}.

\textbf{Inductive} \leq : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \text{Prop} :=
\leq 0 : \forall \, n, \; 0 \leq n
\mid \leq S : \forall \, m \, n, \; m \leq n \rightarrow S \, m \leq S \, n.
```

However, it is propositional:

- constructors are orthogonal
- the induction is structurally justified
- only one, canonical proof of $n \leq m$
Idea: Compile by dependent-pattern matching on the indices

Syntactic criteria for "invertibility":

1. Every non-forced argument of a constructor must be in \( \text{SProp} \): no computational content can escape from \( \text{SProp} \).
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.

\[
\text{Inductive } \leq : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \text{Prop} :=
\leq 0 : \forall n, 0 \leq n
\mid \leq S : \forall m n, m \leq n \rightarrow S m \leq S n.
\]

\[
m \leq n := \text{case}_m \left\{ \begin{array}{c}
0 \rightarrow \leq 0 \\
S m' \rightarrow \text{case}_n \left\{ \begin{array}{c}
0 \rightarrow \bot \\
S n' \rightarrow \leq S (p : m' \leq n')
\end{array} \right. \\
\end{array} \right. 
\]
1. Propositions and equality

2. A universe of strict propositions
   - The \texttt{SProp} universe
   - Semantics & Implementation
The \textit{SProp} universe

\begin{align*}
\text{SProp}_i : \text{Type}_{i+1}
\end{align*}

Predicative (in \textit{Agda}) or impredicative (in \textit{Coq}).

\begin{itemize}
  \item \textit{SProp} enjoys \textit{definitional} proof-irrelevance:
    \begin{align*}
    A : \text{SProp} \quad x, y : A \\
    \hline
    x \equiv y : A
    \end{align*}
  \item The extended theory is independent from UIP or Univalence.
  \item Can encode all types \textit{Prop} except for accessibility.
\end{itemize}
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

\[
\begin{align*}
\text{sEmpty} : \text{SProp} & \quad A : \text{Type} \quad e : \text{sEmpty} \\
\text{sEmpty-elim} e : A
\end{align*}
\]

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

\[
\text{sUnit} : \text{SProp} := \text{False} \rightarrow \text{False}
\]
The **Squash** operation mimicks the truncation operator of HoTT:

\[
\begin{align*}
\text{SquashForm} & \quad A : \text{Type} \\
\text{Squash} & \quad A : \text{SProp} \\
\text{Squash} & \quad A : \text{Type} \quad x : A \\
\text{sq} & \quad x : \text{Squash} A
\end{align*}
\]

\[
\begin{align*}
\text{Unsquash} & \quad A : \text{Type} \quad t : \text{Squash} A \\
P : \text{Squash} A \rightarrow \text{SProp} & \quad e : \forall (a : A), P (\text{sq} a) \\
\text{unsquash} & \quad A P t e : P t
\end{align*}
\]

An inverse **Box** operator can be used to inject \(\text{SProp}\) into \(\text{Type}\) (no \(\text{SProp} \leq \text{Type}\) cumulativity)
Inductive types

- Only allow inductive types whose arguments are all in $\text{SProp}$.
- Otherwise, encode them using fixpoints, $\text{False}$ and $\text{Squash}$.

\[
\leq : \mathbb{N} \to \mathbb{N} \to \text{SProp}
\]

\[
(x, p) \equiv (x, q) : \{ x : \mathbb{N} \mid x \leq y \} \text{ for all } p, q
\]
In Coq: using irrelevance marks on binders for untyped conversion.

In Agda: trivial using type-based conversion
⇒ fixes the unsound irrelevant arguments system

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

Consistency: by a syntactic model in Extensional Type Theory.
⇒ Safe and modular extension with UIP

Independence from UIP: by a syntactic model in HTS/2-level type theory.
⇒ Compatible with univalence
Conclusion

Our new universe of strict propositions:

▶ Fixes issues with \textit{Prop} and singleton eliminations to allow \textit{definitional} irrelevance without introducing axioms.
▶ Eases programming with subset types, decidable propositions (included in \textit{SProp}) and proof-irrelevant types in general.
▶ Is HoTT-compatible: \textit{SProp} models the equality of Bishop sets (as shown by Coquand), a fibrant universe in models.
Open problem: internalization of semi-simplicial types or 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, Capriotti and Kraus):** consider a homotopy type theory inside a theory with uniqueness of identity proofs.

Question: can we avoid introducing a new type theory altogether and use SProp instead?

Partial answer: naively adding a strict equality introduces UIP, rather need a notion of strict sets.