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 `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 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 $\mathrm{id}_a : a =_A a$.
  - 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:

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

Equality is a property (as opposed to a structure)

Extensional Type Theory

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

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

Extensional Type Theory

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

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

- 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!

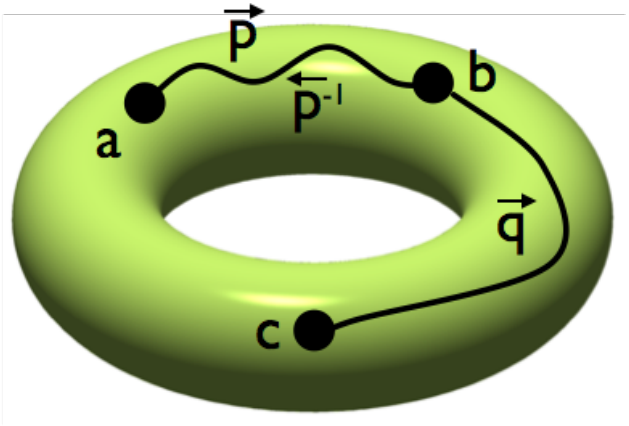
1 Propositions and equality
- Propositions in Homotopy Type Theory
- Propositions in the Calculus of Inductive Constructions
- Extraction and singleton elimination

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



$$
\begin{array}{lcl}
a, b, c & : & \mathbb{T} \\
p & : & a = b \\
\mathsf{id}_a & : & a = a \\
\_^{-1} & : & a = b \to b = a \\
invsym & : & p^{-1} \circ p =_{a=a} id_a
\end{array}
$$

$+$ 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:

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

- 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:

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

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

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

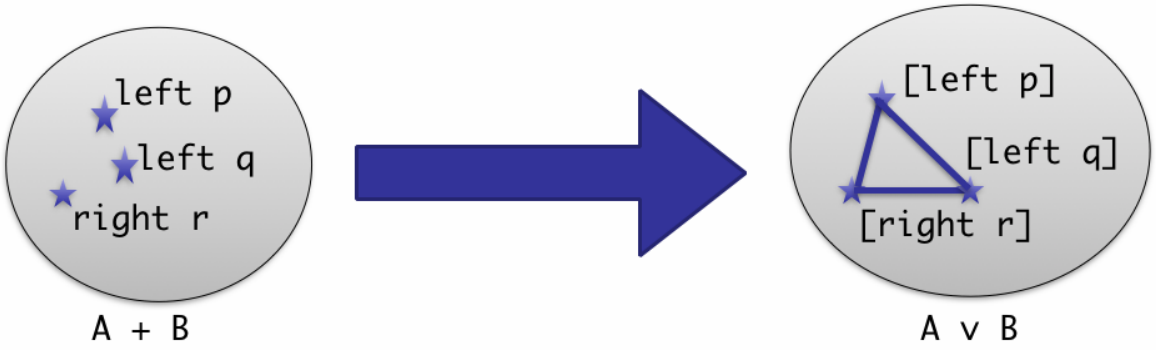
- hSets correspond to ordinary sets, but with a weak equality.
- h-level$(n) \leftrightarrow (n-2)$-truncated type

Truncation

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

$$\textsc{Trunc}\quad \frac{A : \texttt{Type} \qquad x : A}{[x] : \mathsf{Trunc}\ A}$$

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



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:

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

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.
```

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.
- 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...

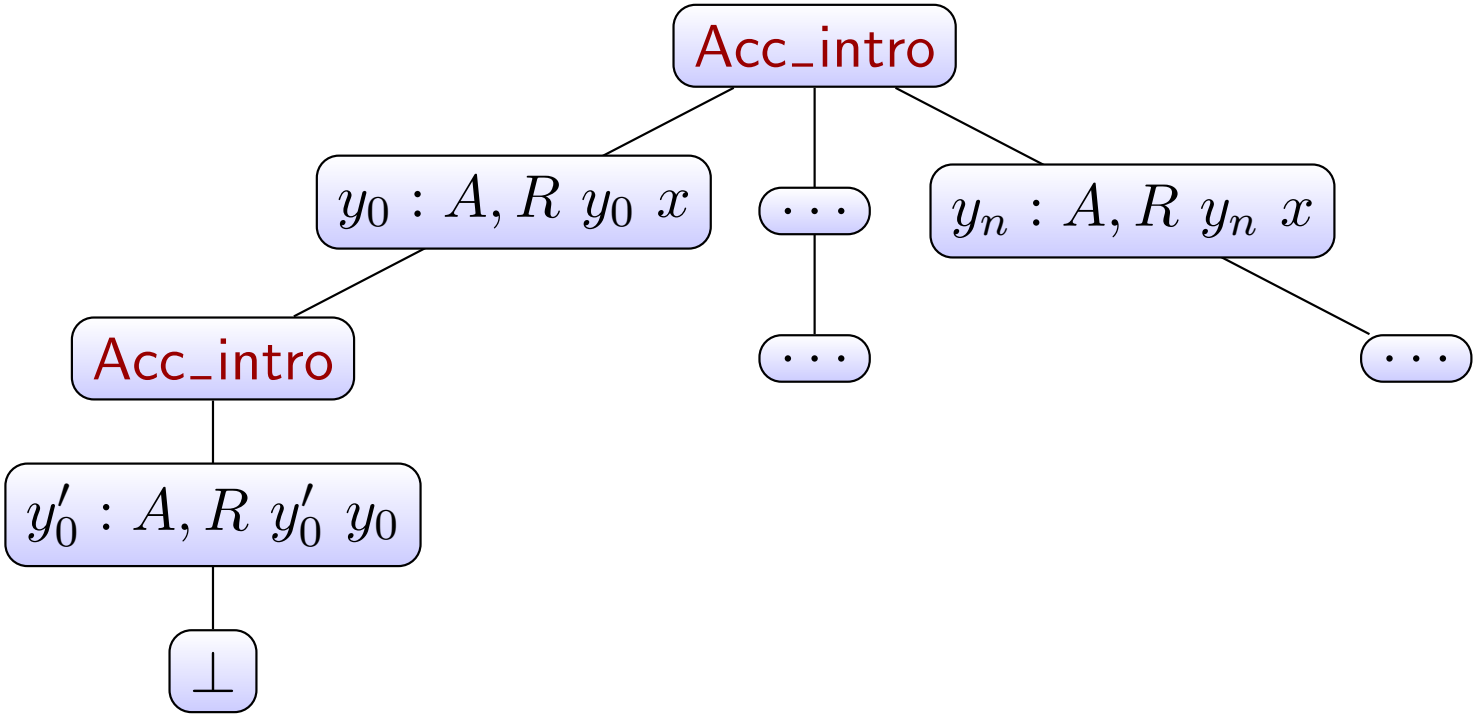
Singleton elimination of equality

- 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 propositions

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 $\underline{n} \leq \underline{m}$

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.

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

$$
m \leq n := \mathsf{case}_m \left\{ \begin{array}{lll} 0 & \mapsto & \leq 0 \\ \mathsf{S}\, m' & \mapsto & \mathsf{case}_n \left\{ \begin{array}{lll} 0 & \mapsto & \bot \\ \mathsf{S}\, n' & \mapsto & \leq \mathsf{S}\, (p : m' \leq n') \end{array} \right\} \end{array} \right\}
$$

1 Propositions and equality

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

The SProp universe

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

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

- SProp enjoys definitional proof-irrelevance:

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

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

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

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

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:

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

$$\textsc{Unsquash}\quad \frac{\begin{array}{c} A : \texttt{Type} \quad t : \mathsf{Squash}\ A \\ P : \mathsf{Squash}\ A \to \texttt{SProp} \quad e : \forall (a : A), P\ (\mathsf{sq}\ a) \end{array}}{\mathsf{unsquash}\ A\ P\ t\ e : P\ t}$$

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

Inductive types

- Only allow inductive types whose arguments are all in $\texttt{SProp}$.
- Otherwise, encode them using fixpoints, False 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 irrrelevance marks on binders for untyped conversion.
- In Agda: trivial using type-based conversion
  $\Rightarrow$ 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.
- Consistency: by a syntactic model in Extensional Type Theory.
  $\Rightarrow$ Safe and modular extension with UIP
- Independence from UIP: by a syntactic model in HTS/2-level type theory.
  $\Rightarrow$ Compatible with univalence

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 (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 models.

Future work

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, Caprioti 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: naïvely adding a strict equality introduces UIP, rather need a notion of strict sets.