# Coq for HoTT

Matthieu Sozeau
Inria Paris & IRIF, Université Paris 7 Diderot

ICMS'16
July 14th 2016
Berlin, Germany

# Coq for Homotopy Type Theory & Univalent Foundations

- Universes: polymorphism, resizing, type-in-type.
- Proof-relevant, dependent equality.

# What are universes?

Universes are the types of *types*, e.g:

- $\mathsf{nat}, \mathsf{bool} : \mathtt{Type}@\{0\}$
- $\mathtt{Type}@\{0\} : \mathtt{Type}@\{1\}$
- $\mathsf{list} : \mathtt{Type}@\{0\} \to \mathtt{Type}@\{0\}$
- $\forall \alpha : \mathtt{Type}@\{0\}, \mathsf{list}\ \alpha : \mathtt{Type}@\{1\}$
- $\forall n : \mathsf{nat}, \{n = 0\} + \{n \neq 0\} : \mathtt{Type}@\{0\}$

## How are they organised?

A *hierarchy* of predicative universes $\texttt{Type}@\{0\} < \texttt{Type}@\{1\} < \ldots$

- Avoids the $\texttt{Type} : \texttt{Type}$ paradox (system $U^{-}$)
- Replicates $\textsc{Russell}$'s paradox of $\{x \mid x \notin x\}$, the set of all sets etc....
- Think of $\texttt{Type@}\{0\}$ as sets, $\texttt{Type@}\{1\}$ as classes etc...

# With polymorphism

Bounded polymorphism:

$$\texttt{Polymorphic Definition}\ \mathrm{id}\ (A : \texttt{Type})\ (a : A) := a$$

$$\underline{\mathrm{id}}_l : \Pi(A : \texttt{Type}@\{l\}),\ A \to A$$

$\Rightarrow l$ is quantified at the definition level now and we can *instantiate* it at each application:

$$\begin{aligned} l < k \vdash_{poly} \underline{\mathrm{id}}_k\ (\Pi(A : \texttt{Type}@\{l\}),\ A \to A)\ \underline{\mathrm{id}}_l \\ : \Pi(A : \texttt{Type}@\{l\}),\ A \to A \end{aligned}$$

# Universes in Coq

Design and implementation choices:

- Keep with Russell's typical ambiguity (i.e. do *inference*)
- But allow user annotations (`Type@{i}`, `id@{Set}`).
- Cumulativity, constraint-based:
  - Unification is complete ($\neq$ algebraic universes), important for automation.
  - Rely on a state-of-the-art constraint checking algorithm (J.-H. Jourdan). (Up to 50% speedups).
- Cumulativity and inductive types (B. Jacobs, A. Timany).

Bender, M.A., Fineman, J.T., Gilbert, S., & Tarjan, R.E. (2011). A new approach to incremental cycle detection and related problems. arXiv:1112.0784.

# Universe variables and constraints

With constraints only, each $i \sqcup j$ is replaced with a fresh $k$ s.t. $i, j \leq k$.

- Generates lot of duplicate universes for the same l.u.b.s

$\Rightarrow$ Minimization / graph reduction (transform $i \leq j$ into $j = i$ when $j$ is not introduced by the user).

Effective strategy:

- HoTT/Coq library
- Category theory library by Timani and Jacobs [FSCD'16] without annotations.

# Going further

- Local and global `type-in-type` options for experimenting. Definitions using type-in-type are flagged/tainted.
- Resizing rules (N. Tabareau, T. Winterhalter)

# Embracing proof-relevant equality

Much work on DTT in Coq focused on propositional equality, assumed proof-irrelevant and using the J eliminator.

An example which rely deeply on equality: A generalized rewriting tactic.

1. Universes
2. Proof-relevant rewriting
   - Generalized rewriting
   - Rewriting with Type-valued relations

# Generalized Rewriting

Why generalized rewriting when we have Id-elimination?

- Id is not the only interesting relation...
- Even with a univalent equality, Id-elim is not enough: *capturing* rewrites under binders.
- Cubical type theory geared towards rewriting based on $\mathrm{ap}$/composition/congruence.

# Rewriting in Type Theory

Moving from substitution to congruence.

- Built-in substitution: Leibniz equality/J-eliminator.
  $\Pi\ A\ (P : A \to \texttt{Type})\ (x\ y : A),\ P\ x \to x = y \to P\ y.$
  - ✓ Applies to any context
  - × Iterated rewrites result in large proof terms: repeats the context that depends on $x$
  - × Restricted to equality, one rewrite at a time

# Rewriting in Type Theory

Moving from substitution to congruence.

- Built-in substitution: Leibniz equality/J-eliminator.
  $\Pi\ A\ (P : A \to \texttt{Type})\ (x\ y : A),\ P\ x \to x = y \to P\ y.$
  - ✓ Applies to any context
  - × Iterated rewrites result in large proof terms: repeats the context that depends on $x$
  - × Restricted to equality, one rewrite at a time
- Congruence.
  $\mathrm{ap} : \Pi\ A\ B\ (f : A \to B)\ (x\ y : A),\ x = y \to f\ x = f\ y$
  - × Applies at the toplevel only
  - ✓ Smaller proof term: mentions the changed terms only
  - ✓ Generalizes to n-ary, parallel rewriting
  - × Still restricted to equality

# Rewriting in Type Theory

Moving from substitution to congruence.

- Built-in substitution: Leibniz equality/J-eliminator.
  $\Pi\ A\ (P : A \to \texttt{Type})\ (x\ y : A),\ P\ x \to x = y \to P\ y$.
  - ✓ Applies to any context
  - × Iterated rewrites result in large proof terms: repeats the context that depends on $x$
  - × Restricted to equality, one rewrite at a time
- Congruence.
  $\mathrm{ap} : \Pi\ A\ B\ (f : A \to B)\ (x\ y : A),\ x = y \to f\ x = f\ y$
  - × Applies at the toplevel only
  - ✓ Smaller proof term: mentions the changed terms only
  - ✓ Generalizes to n-ary, parallel rewriting
  - × Still restricted to equality

One can build a set of combinators to rewrite in depth: HOL conversions [Paulson 83], ELAN strategies, `rewrite_strat` tactic in Coq.

# Basic idea

Apply congruence rules (i.e. lemmas of type Proper) or the rewrite lemma to produce a new goal.

```
Class Proper {A : Type} (R : relation A) (m : A) :=
  proper : R m m.

Definition respectful {A B} (R : relation A) (R' : relation B)
: relation (A → B) :=
 fun (f g : A → B) ⇒ ∀ x y : A, R x y → R' (f x) (g y).

Example proper_id A (RA : relation A) :
  Proper (respectful RA RA) (@id A).

Check proper_id : ∀ A RA, Proper (RA ++> RA) (@id A).
```

# Example

```
Inductive ex {A : Type} (P : A → Prop) : Prop :=
    ex_intro : ∀ x : A, P x → ex P.

Instance ex_iff_P A :
  Proper (pointwise_relation A iff ++> iff) (@ex A).
```

# Example

```
Inductive ex {A : Type} (P : A → Prop) : Prop :=
    ex_intro : ∀ x : A, P x → ex P.

Instance ex_iff_P A :
  Proper (pointwise_relation A iff ++> iff) (@ex A).

  A : Type
  P, Q : A → Prop
  H : ∀ x : A, P x ↔ Q x
  HnP : ∃ x : A, ¬ P x
  ============================
   ∃ x : A, ¬ Q x

  setoid_rewrite ← H. exact HnP.
Qed.
```

# Generalized Rewriting in Type Theory

D. Basin [NuPRL, 94], C. Sacerdoti Coen [Coq, 04], Sozeau [Coq, 09]

- Generalized to any relation
  Proper $(\mathrm{iff} \Longrightarrow \mathrm{iff})\ \mathrm{not} \triangleq \Pi\ P\ Q,\ P \leftrightarrow Q \to \neg\ P \leftrightarrow \neg\ Q$
- Multiple signatures for a given constant
  Proper $(\mathrm{impl} \longrightarrow \mathrm{impl})\ \mathrm{not}$

D. Basin [NuPRL, 94], C. Sacerdoti Coen [Coq, 04], Sozeau [Coq, 09]

- Generalized to any relation
  Proper $(\mathrm{iff} \Longrightarrow \mathrm{iff})\ \mathrm{not} \triangleq \Pi\ P\ Q,\ P \leftrightarrow Q \to \neg\ P \leftrightarrow \neg\ Q$
- Multiple signatures for a given constant
  Proper $(\mathrm{impl} \longrightarrow \mathrm{impl})\ \mathrm{not}$

Requires proof search:

- Heuristic in NuPRL based on subrelations $(\mathrm{impl} \subset \mathrm{iff})$
- Complete procedure in Coq.

# Proper instances

- Algebraic presentation of signatures, supporting higher-order functions and polymorphism:

$\Pi\ A\ B\ C\ R_0\ R_1\ R_2,$
Proper $((R_1 \Longrightarrow R_2) \Longrightarrow (R_0 \Longrightarrow R_1) \Longrightarrow (R_0 \Longrightarrow R_2))$
$\quad$ (@compose $A\ B\ C$)

# Proper instances

- Algebraic presentation of signatures, supporting higher-order functions and polymorphism:

  $\Pi\ A\ B\ C\ R_0\ R_1\ R_2,$
  $\mathsf{Proper}\ ((R_1 \Longrightarrow R_2) \Longrightarrow (R_0 \Longrightarrow R_1) \Longrightarrow (R_0 \Longrightarrow R_2))$
  $\quad (\texttt{@}\mathrm{compose}\ A\ B\ C)$

- Extensible signatures (shallow embedding)

  $\mathrm{all} : \forall\ A : \texttt{Type}, (A \to \texttt{Prop}) \to \texttt{Prop}$

  $\Pi\ A, \mathsf{Proper}\ (\mathrm{pointwise\_relation}\ A\ \mathrm{iff} \Longrightarrow \mathrm{iff})\ (\texttt{@}\mathrm{all}\ A)$

# Applications

- "Poor-man"'s quotients: reasoning on setoids (type + equivalence relation)
- "Poor-man"'s functional extensionality
- Bisimulations
- Rewriting with order relations, e.g. set inclusion.

# Impredicativity helped

All fine with relations in `Prop`, how about `Type@{i}`-valued relations?

$$\text{Proper} : \Pi A : \texttt{Type}@\{i\}, (A \to A \to \texttt{Type}@\{j\}) \to A \to \texttt{Type}@\{j\}.$$

Need to show, under $A : \texttt{Type}@\{i\}$:

$$\begin{array}{ll} \text{Proper} & ((A \to A \to \texttt{Type}@\{j\}) \to A \to \texttt{Type}@\{j\}) \\ & (\mathrm{iso\_rel}\ A \Longrightarrow \mathrm{eq}\ A \Longrightarrow \mathrm{iso}) \\ & (\text{Proper}\ A) \end{array}$$

Requires: $\texttt{Type}@\{\max(i, j+1)\} \leq \texttt{Type}@\{i\}$ i.e. $j < i$.
But then $\mathrm{iso}\ A : \texttt{Type}@\{i\} \not\leq \texttt{Type}@\{j\} \Rightarrow$ inconsistency.

# Universe Polymorphism

With universe polymorphism:

$\text{Proper}_{i\,j} : \Pi A : \texttt{Type}@\{i\}, (A \to A \to \texttt{Type}@\{j\}) \to A \to \texttt{Type}@\{j\}$

We can show, under $A : \texttt{Type}@\{i\}$:

$$\begin{array}{ll} \text{Proper}_{i'\,j'} & ((A \to A \to \texttt{Type}@\{j\}) \to A \to \texttt{Type}@\{j\}) \\ & (\mathrm{iso\_rel}\ A \Longrightarrow \mathrm{eq}\ A \Longrightarrow \mathrm{iso}) \\ & (\text{Proper}_{i\,j}\ A) \end{array}$$

The constraint $\max(i, j+1) \leq i'$ is satisfiable.

Actually, $\mathrm{crelation}(A : \texttt{Type}@\{i\}) := A \to A \to \texttt{Type}@\{j\}$ is already problematic: no relation equivalence or subrelation definition possible.

# Going further: heterogeneous relations and transfer

Rewriting is just one instance of reasoning up to monotonicity/ logical relations.

- **Monotonicity**: `coqrel` library (J. Koenig, CoqPL'16), for simulation proofs in OS verification.
- **Transfer**: library to transfer theorems along isomorphisms (T. Zimmermann): e.g. from iso nat N, transfer nat_ind to N_ind.

```
Check nat_ind
      : ∀ P : nat → Prop,
        P 0 → (∀ n : nat, P n → P (S n)) → ∀ n : nat, P n.
```

Both use proof-search, similar setup as generalized rewriting.

# Rewriting with dependencies and heterogeneous relations

- Relations become heterogeneous (e.g. relating a nat to an $\mathbb{N}$)
- Relations become dependent (relating dependent products)

New central definition:

```
Class Related A B (R : A → B → Type) (m : A) (n : B) :=
  related : R m n.
Notation Proper R m := (Related _ _ R m m).
```

# Rewriting with dependencies and heterogeneous relations

Generalize the logical relation to dependent products:

$$R \Longrightarrow S : \mathrm{relation}(A \to B) \triangleq \lambda f g, \forall x\, y, R\, x\, y \to S\, (f\, x)\, (g\, y)$$

Becomes:

$$\begin{aligned}
&\forall (A\ B : \mathtt{Type})(C : A \to \mathtt{Type})(D : B \to \mathtt{Type})\\
&(R : A \to B \to \mathtt{Type})\\
&(S : \forall\ x\ y\ (\alpha : R\ x\ y), C\ x \to D\ y \to \mathtt{Type})\\
&: (\Pi x : A.C) \to (\Pi x : B.D) \to \mathtt{Type} \triangleq\\
&\lambda f g, \forall\ x\ y\ (e : R\ x\ y), S\ x\ y\ \alpha\ (f\ x)(g\ y)
\end{aligned}$$

Notation (from J. Koenig's `coqrel` library):

$$\forall \alpha : R\ x\ y, S$$

# Example morphisms

$\mathsf{cons} : \forall A : \mathtt{Type}, A \to \mathsf{list}\ A \to \mathsf{list}\ A$

$\mathsf{Proper}(\forall \alpha : \mathsf{Equiv}\ A\ B, \alpha_R \Longrightarrow \mathsf{list}_{\mathsf{eq}}\ \alpha_{\mathsf{R}} \Longrightarrow \mathsf{list}_{\mathsf{eq}}\ \alpha_R)(@cons)$

where $\alpha_R \triangleq \lambda\ x\ y, \mathrm{equiv}\ \alpha\ x = y$

```
Definition EquivRel {A B} (I : Equiv A B) : A → B → Prop
  := fun (p : A) (q : B) ⇒ equiv p = q.
Lemma equiv_all_iff : ∀ A B (E : Equiv A B),
  Related ((EquivRel E ⟹ iff) ⟹ iff) (@all A) (@all B).
```

# Dependent rewriting

$nat$ : `Type`
$zero$ : $nat$
$eq$ : $nat \to nat \to$ `Prop`
$div$ : $nat \to \forall\ m : nat, \text{nonzero}\ m \to nat$
$n, m, m' : nat$
$e : eq\ m\ m'$
$pn : \text{nonzero}\ m$
$pn' : \text{nonzero}\ m'$
============================
$eq\ (div\ n\ m\ pn)\ (div\ n\ m'\ pn')$

`rewrite` $e$. `reflexivity`.
`Qed`.

# Applications

- Rewriting with Equiv, *Id* in `Type@{.}`
- Computational relations (e.g. apartness of reals in `CoRN`)
- Transfer/reasoning modulo isomorphisms.

# The end

Thanks for your attention