Coq for HoTT

Matthieu Sozeau
Inria Paris & IRIF, Université Paris 7 Diderot

ICMS’16
July 14th 2016
Berlin, Germany
Universes: polymorphism, resizing, type-in-type.

Proof-relevant, dependent equality.
What are universes?

Universes are the types of \textit{types}, e.g:

\begin{itemize}
  \item $\text{nat, bool} : \text{Type@\{0\}}$
  \item $\text{Type@\{0\}} : \text{Type@\{1\}}$
  \item $\text{list} : \text{Type@\{0\}} \rightarrow \text{Type@\{0\}}$
  \item $\forall \alpha : \text{Type@\{0\}}, \text{list } \alpha : \text{Type@\{1\}}$
  \item $\forall n : \text{nat}, \{n = 0\} + \{n \neq 0\} : \text{Type@\{0\}}$
\end{itemize}
How are they organised?

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

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

Bounded polymorphism:

**Polymorphic Definition** \( \text{id} (A : \text{Type}) (a : A) := a \)

\( \text{id}_l : \Pi(A : \text{Type}@\{l\}), \ A \rightarrow A \)

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

\[ l < k \vdash_{poly} \text{id}_k (\Pi(A : \text{Type}@\{l\}), \ A \rightarrow A) \text{id}_l \]

\( : \Pi(A : \text{Type}@\{l\}), \ A \rightarrow A \)
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).

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

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

⇒ 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 \texttt{Id}-elimination?

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

Moving from substitution to congruence.

- Built-in substitution: Leibniz equality/J-eliminator.
  \[ \Pi A (P : A \rightarrow \text{Type}) (x y : A), P x \rightarrow x = y \rightarrow P y. \]
  
  ✓ 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.
  \[ \text{ap} : \Pi A B (f : A \rightarrow B) (x y : A), x = y \rightarrow f x = f y. \]
  
  ✓ 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 tactic in Coq.
Rewriting in Type Theory

Moving from substitution to congruence.

- **Built-in substitution:** Leibniz equality/J-eliminator.
  \[ \Pi A \ (P : A \to \text{Type}) \ (x \ y : A), \ P \ x \to x = y \to P \ y. \]
  - ✓ 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.**
  \[ \text{ap} : \Pi A \ B \ (f : A \to B) \ (x \ y : A), \ x = y \to f \ x = f \ y \]
  - ✗ 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.
\[ \Pi \ A \ (P : A \to \text{Type}) \ (x \ y : A), \ P \ x \to x = y \to P \ y. \]

✓ 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.
\[ \text{ap} : \Pi \ A \ B \ (f : A \to B) \ (x \ y : A), \ x = y \to f \ x = f \ y \]

✗ 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** 
\[
\{ A : \text{Type} \} \ ( R : \text{relation} \ A ) \ ( m : A ) := \\
\text{proper} : R \ m \ m.
\]

**Definition** respectful 
\[
\{ A B \} \ ( R : \text{relation} \ A ) \ ( R' : \text{relation} \ B ) : \text{relation} \ ( A \rightarrow B ) := \\
\text{fun} \ ( f \ g : A \rightarrow B ) \Rightarrow \forall \ x \ y : A, R \ x \ y \rightarrow R' ( f \ x ) ( g \ y ).
\]

**Example** proper_id \( A \) \( RA : \text{relation} \ A \) :
Proper (respectful \( RA \ RA \))(@id \( A \)).

**Check** proper_id : \( \forall \ A \ RA, \) Proper (\( RA \leftrightarrow RA \))(@id \( A \)).
Inductive ex \( \{ A : Type \} (P : A \to Prop) : Prop \) :=
\[ \text{ex_intro} : \forall x : A, P x \to \text{ex } P. \]

Instance ex_iff_P A :
\[ \text{Proper (pointwise_relation } A \text{ iff } \leftrightarrow \text{ iff) } (@\text{ex } A). \]
Example

Inductive ex \{ A : Type \} (P : A \to Prop) : Prop :=
  ex_intro : \forall x : A, P x \to ex P.

Instance ex_iff_P A :
  Proper (pointwise_relation A iff \leftrightarrow iff) (@ex A).

A : Type
P, Q : A \to Prop
H : \forall x : A, P x \leftrightarrow Q x
HnP : \exists x : A, \neg P x

\exists x : A, \neg Q x

setoid_rewrite \leftarrow H. exact HnP.
Qed.
D. Basin [NuPRL, 94], C. Sacerdoti Coen [CoQ, 04], Sozeau [CoQ, 09]

- Generalized to any relation
  \(\text{Proper } (\text{iff } \leftrightarrow \text{iff}) \not\equiv \Pi P Q, P \leftrightarrow Q \rightarrow \neg P \leftrightarrow \neg Q\)

- Multiple signatures for a given constant
  \(\text{Proper } (\text{impl } \rightarrow \text{impl}) \not\equiv\)
D. Basin [NuPRL, 94], C. Sacerdoti Coen [CoQ, 04], Sozeau [CoQ, 09]

- Generalized to any relation
  \[ \text{Proper (iff } \iff \text{ iff) not } \triangleq \Pi \ P \ Q, \ P \leftrightarrow Q \rightarrow \neg P \leftrightarrow \neg Q \]

- Multiple signatures for a given constant
  \[ \text{Proper (impl } \implies \text{ impl) not} \]

Requires proof search:

- Heuristic in NuPRL based on subrelations \( \text{impl } \subset \text{ iff} \)
- Complete procedure in CoQ.
Proper instances

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

\[
\Pi A B C R_0 R_1 R_2, \\
\text{Proper (}(R_1 \to R_2) \to (R_0 \to R_1) \to (R_0 \to R_2)) \\
(\@\text{compose } A B C)
\]
Proper instances

- Algebraic presentation of signatures, supporting higher-order functions and polymorphism:
  \[
  \Pi A B C R_0 R_1 R_2, \\
  \text{Proper } ((R_1 \leftrightarrow R_2) \leftrightarrow (R_0 \leftrightarrow R_1) \leftrightarrow (R_0 \leftrightarrow R_2)) \\
  \quad \text{(\text{@compose } A B C)}
  \]

- Extensible signatures (shallow embedding)
  \[
  \text{all} : \forall A : \text{Type}, (A \to \text{Prop}) \to \text{Prop} \\
  \Pi A, \text{Proper } (\text{pointwise_relation } A \iff \leftrightarrow \iff) \text{ (\text{@all } A)}
  \]
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.
1 Universes

2 Proof-relevant rewriting
   - Generalized rewriting
   - Rewriting with Type-valued relations
Impredicativity helped

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

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

Need to show, under \( A : \text{Type}@\{i\} \):

\[
\text{Proper} \quad ((A \to A \to \text{Type}@\{j\}) \to A \to \text{Type}@\{j\}) \\
(iso\_rel A \implies eq A \implies iso) \\
(Proper A)
\]

Requires: \( \text{Type}@\{\max(i, j + 1)\} \leq \text{Type}@\{i\} \) i.e. \( j < i \).

But then \( \text{iso} A : \text{Type}@\{i\} \not\leq \text{Type}@\{j\} \implies \text{inconsistency} \).
With universe polymorphism:

\[ \text{Proper}_{i,j} : \Pi A : \text{Type}_{i}, (A \to A \to \text{Type}_{j}) \to A \to \text{Type}_{j} \]

We can show, under \( A : \text{Type}_{i} \):

\[ \text{Proper}_{i',j'} ((A \to A \to \text{Type}_{j}) \to A \to \text{Type}_{j}) (\text{iso\_rel } A \implies \text{eq } A \implies \text{iso}) (\text{Proper}_{i,j} A) \]

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

Actually, \( \text{crelation}(A : \text{Type}_{i}) := A \to A \to \text{Type}_{j} \) 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`

```
: \ A P : nat \rightarrow Prop,
P 0 \rightarrow (\ A n : nat, P n \rightarrow P (S n)) \rightarrow \ A 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 `N`)
- Relations become dependent (relating dependent products)

New central definition:

```coq
Class Related A B (R : A → B → Type) (m : A) (n : B) := related : R m n.
Notation Proper R m := (Related _ _ R m m).
```
Generalize the logical relation to dependent products:

\[ R \iff S : \text{relation}(A \rightarrow B) \triangleq \lambda f g, \forall x y, R x y \rightarrow S (f x)(g y) \]

Becomes:

\[ \forall (A B : \text{Type})(C : A \rightarrow \text{Type})(D : B \rightarrow \text{Type}) \]
\[ (R : A \rightarrow B \rightarrow \text{Type}) \]
\[ (S : \forall x y (\alpha : R x y) C x \rightarrow D y \rightarrow \text{Type}) \]
\[ : (\Pi x : A . C) \rightarrow (\Pi x : B . D) \rightarrow \text{Type} \triangleq \lambda f g, \forall x y (e : R x y), S x y \alpha (f x)(g y) \]

Notation (from J. Koenig’s coqrel library):

\[ \forall \alpha : R x y, S \]
Example morphisms

\[
\text{cons} : \forall A : \text{Type}, \ A \to \text{list} \ A \to \text{list} \ A
\]

Proper(\(\forall \alpha : \text{Equiv} \ A \ B, \ \alpha_R \implies \text{list}_{eq} \ \alpha_R \implies \text{list}_{eq} \ \alpha_R\))(\text{@cons})

where \(\alpha_R \triangleq \lambda x \ y, \ \text{equiv} \ \alpha \ x = y\)

Definition \text{EquivRel} \ \{A \ \ B\ \} \ (I : \ \text{Equiv} \ A \ B) : \ A \to \ B \to \text{Prop}

\[:= \text{fun} \ (p : A) \ (q : B) \Rightarrow \text{equiv} \ p = q.\]

Lemma \text{equiv\_all\_iff} : \ \forall \ A \ B \ (E : \ \text{Equiv} \ A \ B), \ \text{Related} \ ((\text{EquivRel} \ E \implies \iff) \implies \iff) \ (@\text{all} \ A) \ (@\text{all} \ B).\]
Dependent rewriting

\begin{align*}
nat & : \text{Type} \\
zero & : nat \\
eq & : nat \to nat \to \text{Prop} \\
\text{div} & : nat \to \forall m : nat, \text{nonzero } m \rightarrow nat \\
n, m, m' & : nat \\
e & : eq m m' \\
\text{pn} & : \text{nonzero } m \\
\text{pn}' & : \text{nonzero } m' \\
\text{eq} (\text{div} n m \text{ pn})(\text{div} n m' \text{ pn}') & \\
\text{rewrite } e. \text{ reflexivity.} \\
\text{Qed.}
\end{align*}
Applications

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