Congruence in univalent type theory

Luis Scoccola
lscoccol@uwo.ca

University of Western Ontario

June 11, 2019

Goals for the talk

- Congruence and congruence closure for propositional equality.
- Solution of Selsam & de Moura for a non-univalent type theory.
- Proposed approach for congruence in the univalent case.
- Issues with congruence closure.

Informal definitions

Definition

A relation R satisfies **congruence** *if* $\forall f$

$$x_i \, R \, y_i \text{ for } 0 \leq i \leq n \Rightarrow f(x_1, \ldots, x_n) \, R \, f(y_1, \ldots, y_n).$$

The **congruence closure** *of R is the smallest equivalence relation that satisfies congruence and contains R.*

Example

During a proof, determine whether $x = y$ follows from applying **reflexivity**, **symmetry**, **transitivity**, or **congruence lemmas** to the equalities in our context.

Congruence in dependent type theory

Propositional equality is an equivalence relation:

$$\begin{aligned} \mathsf{refl} &: x = x \\ \mathsf{inv} &: x = y \to y = x \\ \mathsf{concat} &: x = y \to y = z \to x = z \end{aligned}$$

Can also prove **congruence lemma** (for non-dependent function):

$$\mathsf{congr}_f : (f : A \to B) \to (x =_A y) \to f(x) =_B f(y).$$

But if $f : (a : A) \to B(a)$, the above doesn't type check.

A solution: Heterogeneous equality

Definition

Heterogeneous equality *is the inductive family*

$$\mathsf{heq} : (A, A' : \mathcal{U}) \to A \to A' \to \mathcal{U}$$

generated by $\mathsf{refl} : (A : \mathcal{U}) \to (a : A) \to \mathsf{heq}(a, a)$.

Selsam & de Moura implement full congruence closure procedure in Lean 3 using heq.

Need to assume an axiom to prove the congruence lemmas:

$$\texttt{ofheq} : (A : \mathcal{U}) \to (x, y : A) \to \mathsf{heq}(x, y) \to x =_A y.$$

The axiom `ofheq` implies that paths in $\mathcal{U}$ transport trivially:

$$(e : A =_{\mathcal{U}} A) \to a = \mathsf{transport}^{X \mapsto X}(e, a).$$

$\mathcal{U}$ can't be univalent.

Congruence in univalent type theory?

Use **pathovers**.

Definition

Given a type $B : \mathcal{U}$ *and a type family* $X : B \to \mathcal{U}$*, the type family*

$$\mathsf{pathover} : (b, b' : B) \to (b = b') \to X(b) \to X(b') \to \mathcal{U}$$

is defined by path induction.

We write $x =^{B}_{\langle e \rangle} x'$ for $\mathsf{pathover}(b, b', e, x, x')$.

The congruence lemma

I will describe an inductive algorithm that produces:

- A pathover type for each dependent family.
- A congruence lemma for each dependent function.

This is implemented as a tactic in Lean 3.

We have to be careful with one thing:
Congruence lemmas depend on previous congruence lemmas.

The congruence lemma (cont.)

Example

Congruence lemma for

$$\mathsf{cons} : (n : \mathbb{N}) \to A \to \mathsf{vec}_A(n) \to \mathsf{vec}_A(\mathsf{succ}(n)),$$

should have type

$$\begin{aligned} \mathsf{congr}_{\mathsf{cons}} \ & (n, m : \mathbb{N})\ (x, y : A)\ (xs : \mathsf{vec}_A(n))\ (ys : \mathsf{vec}_A(m)) \\ & (e_1 : n = m)\ (e_2 : x = y)\ (e_3 : xs =_{\langle e_1 \rangle} ys) : \\ & \mathsf{cons}(n, x, xs) =^{\mathsf{vec}_A}_{\langle \mathsf{congr}_{\mathsf{succ}}(e_1) \rangle} \mathsf{cons}(m, y, ys) \end{aligned}$$

where $\mathsf{congr}_{\mathsf{succ}} : (n, m : \mathbb{N}) \to (n = m) \to \mathsf{succ}(n) = \mathsf{succ}(m)$.

The algorithm

Given context Γ, and dependent function

$$h : (x_0 : A_0) \to (x_1 : A_1(x_0)) \to \cdots \to (x_n : A_n(x_1, \ldots, x_{n-1})) \\ \to A_{n+1}(x_1, \ldots, x_n),$$

in context Γ.

(1) **Decompose the type of h as type families applied to dependent functions:** write $A_i(\overline{x}_{i-1}) \equiv C_i(\overline{f}_i(\overline{x}_{i-1}))$ such that

- C_i is not an application;
- $\overline{f}_i$ is a sequence $f_i^1, \ldots, f_i^{k(i)}$ of dependent functions.

The algorithm (cont.)

(2) **Define the pathovers for the type families C_i:**

$$\mathsf{Id}_{C_i} :\equiv \lambda \overline{x}_i, \overline{x}'_i, \overline{e}_i, c, c'.\mathsf{congr}_{C_i}(\overline{x}_i, \overline{x}'_i, \overline{e}_i)_* c = c'$$

(3) **Define the congruence for all the functions f_i^k:** Recursively. Caveat: Each function might be a composite, so return the composite of the congruences.

The algorithm (cont.)

(4) **Define the congruence for h:**

$$\begin{aligned}
\mathsf{congr}_h : (x_0 : C_0) &\to (x_1 : C_1(\overline{f}_1(\overline{x}_0))) \to \cdots \to (x_n : C_n(\overline{f}_n(\overline{x}_{n-1}))) \\
(x'_0 : C_0) &\to (x'_1 : C_1(\overline{f}_1(\overline{x}'_0))) \to \cdots \to (x'_n : C_n(\overline{f}_n(\overline{x}'_{n-1}))) \\
(e_0 : \mathsf{Id}_{C_0}) &\to (e_1 : \mathsf{Id}_{C_1}(\mathsf{congr}_{\overline{f}_1}(\overline{e}_0)) \to \cdots \\
&\to (e_n : \mathsf{Id}_{C_n}(\mathsf{congr}_{\overline{f}_n}(\overline{e}_{n-1})) \\
&\to \mathsf{Id}_{C_{n+1}}(\mathsf{congr}_{\overline{f}_{n+1}}(\overline{e}_n)),
\end{aligned}$$

By path induction on the pathovers $e_0, \cdots, e_n$, using refl.

Congruence lemma

Also gives us a useful characterization of the identity types of:

- structures;
- iterated sigmas.

Congruence closure

Work in progress:

- Must keep a congruence data structure for each type family.
- Coherence problems (e.g. congruence of concatenation is concatenation of congruences).
- Some of them can be dealt with by working in a cubical type theory (e.g. composite of congruences is definitionally equal to congruence of composites, inverse of inverse is identity).
- Different equalities between the same pair of elements: cannot use union-find data structure for congruence closure. Use graphs instead, but this is inefficient.

These are not problems if we limit congruence closure to type families that depend on sets.

Thank you for listening!