CONGRUENCE IN UNIVALENT TYPE THEORY

LUIS SCOCCOLA

Introduction. The main purpose of congruence closure procedures is to automate the application of basic properties of equality, such as transitivity and congruence (i.e., $x = y$ implies $f(x) = f(y)$). The only current implementation of a full congruence closure procedure for intensional type theory, Selsam & de Moura (IJCAR 2016), adds an axiom to the type theory that is inconsistent with univalence. This axiom is used when proving the congruence lemmas. We describe an approach for automatically synthesizing congruence lemmas that is compatible with univalence.

Congruence using heterogeneous equality. The main difficulty with congruence in dependent type theory is that the "obvious" congruence lemmas don't type check. For example, given $f : (a : A) \to B(a)$, the expression $\mathsf{congr}_f : (a, a' : A) \to (a = a') \to f(a) = f(a')$ doesn't type check, since $f(a) : B(a)$ whereas $f(a') : B(a')$. One way to fix this, is to use heterogeneous equalities, a weakening of McBride's "John Major equality".

Definition 1. ***Heterogeneous equality*** *is defined as an inductive family*

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

with one constructor $\mathsf{refl}^{\mathsf{heq}} : (A : \mathcal{U}) \to (a : A) \to \mathsf{heq}(A, A, a, a)$.

One can use this to write congruence lemmas that type check, $\mathsf{congr}_f : (a, a' : A) \to \mathsf{heq}(A, A, a, a') \to \mathsf{heq}(B(a), B(a'), f(a), f(a'))$, but these cannot be proven without modifying the type theory. The solution of Selsam & de Moura is to assume the following *axiom*

$$\mathtt{ofheq} : (a, a' : A) \to \mathsf{heq}(A, A, a, a') \to a = a'.$$

Using this axiom, they prove a congruence lemma $\mathtt{hcongr}_n$ for each $n \geq 1$. The idea is that $\mathtt{hcongr}_n$ is the congruence lemma for dependent functions with n arguments.

Incompatibility with univalence. Recall, from [2], the pathover type family.

Definition 2 (Licata & Brunerie (LICS 2015))**.** *Given a type* $B : \mathcal{U}$ *and a type family* $X : B \to \mathcal{U}$*, define the type family*

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

by path induction. We denote the type $\mathsf{pathover}_B(b, b', e, a, a')$ *by* $a =^B_{\langle e \rangle} a'$.

The types heq and $\mathsf{pathover}$ are related as follows.

Lemma 3. *For any* $a : A$ *and* $a' : A'$ *we have*

$$\mathsf{heq}(A, A', a, a') \simeq \sum_{e : A = A'} \mathsf{pathover}_{\mathsf{Id} : \mathcal{U} \to \mathcal{U}}(A, A', e, a, a').$$

From this, it follows that the axiom $\mathtt{ofheq}$ implies UIP (uniqueness of identity proofs), and thus, that it is inconsistent with univalence.

Congruence using pathover. In order to synthesize congruence lemmas for type families with arbitrarily many parameters, we must define a pathover type for each such family.

It is conceptually clearer to describe this generalization, and the congruence lemmas, in terms of the category of contexts of our type theory. Given a context Γ and inhabitants $a, b : \Gamma$, one can define an equality context $a = b$ by context induction and path induction ([1, Proposition 3.3.1]). Similarly, given a context extension $\Gamma.\Gamma'$, inhabitants $a, b : \Gamma$, $a' : \Gamma'(a)$, $b' : \Gamma'(b)$, and an equality $e : a = b$, one can define a context of pathovers $a' =_{\langle e \rangle} b'$.

Now, given two context extensions $\Gamma.\Gamma'$ and $\Delta.\Delta'$ and a map $f.f' : \Gamma.\Gamma' \to \Delta.\Delta'$ between them, we can use path induction to prove the following congruence lemma

$$a, b : \Gamma, a' : \Gamma'(a), b' : \Gamma'(b), e_1 : a = b, e_2 : a' =_{\langle e_1 \rangle} b' \vdash$$
$$\mathsf{congr}_{f'}(a, b, a', b', e_1, e_2) : f'(a, a') =_{\langle \mathsf{congr}_f(a,b,e_1) \rangle} f'(b, b').$$

Notice how the type of the congruence lemma for f' uses the congruence lemma for f.

Main result 4. *We give an algorithm to automatically state and prove congruence lemmas for any dependent function.*

The main complication is in correctly characterizing the identity types of contexts. This becomes apparent in the following example.

Example 5. The congruence lemma for $\mathsf{cons} : (n : \mathbb{N}) \to A \to \mathsf{vec}_A(n) \to \mathsf{vec}_A(\mathsf{succ}(n))$ is

$$\mathsf{congr}_{\mathsf{cons}}(n, m, x, y, xs, ys, e_1, e_2, e_3) : \mathsf{cons}(n, x, xs) =_{\langle \mathsf{congr}_{\mathsf{succ}}(e_1) \rangle} \mathsf{cons}(m, y, ys)$$

where $e_1 : n = m$, $e_2 : x = y$, $e_3 : xs =_{\langle e_1 \rangle} ys$, and $\mathsf{congr}_{\mathsf{succ}} : (n, m : \mathbb{N}) \to (n = m) \to \mathsf{succ}(n) = \mathsf{succ}(m)$. We see that, although cons takes as input $x : A$, the type of its codomain does not depend on x, and thus the pathover returned by its congruence lemma should not live over the path $e_2 : x = y$.

This means that we need a representation of contexts that takes dependency into account. We represent contexts as **inverse diagrams**. This is the final ingredient in the procedure. Since the description of the full procedure requires some setting up, we illustrate how it works with an example.

Example 6. We first identify the domain and codomain contexts of cons, and represent them as inverse diagrams

$$A \quad \begin{array}{c} \mathsf{vec}_A \\ \downarrow \\ \mathbb{N}, \end{array} \qquad \begin{array}{c} \mathsf{vec}_A \\ \downarrow \\ \mathbb{N} \end{array}$$

Analyzing the type of cons, we see that cons lives over the context morphism succ

$$\begin{array}{ccc} (n : \mathbb{N}).(x : A, xs : \mathsf{vec}_A(n)) & \xrightarrow{\mathsf{succ.cons}} & (n : \mathbb{N}).(xs : \mathsf{vec}_A(n)) \\ \downarrow & & \downarrow \\ (n : \mathbb{N}) & \xrightarrow{\qquad\qquad \mathsf{succ} \qquad\qquad} & (n : \mathbb{N}). \end{array}$$

So, inductively, we produce the congruence lemma for succ

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

Finally, we use path induction, and induction on the inverse diagrams, to correctly characterize the identity types of the domain and codomain of cons. Giving, for example,

$$(e_1 : n = m, e_2 : x = y, e_3 : xs =_{\langle e_1 \rangle} ys)$$

for the domain. Putting these things together, we get the congruence lemma of Example 5.

References

[1] Richard Garner. "Two-dimensional models of type theory". In: *Mathematical Structures in Computer Science* 19.4 (2009), 687–736. DOI: 10.1017/S0960129509007646.

[2] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study, Princeton, NJ, 2013. URL: http://homotopytypetheory.org/book.