CONGRUENCE CLOSURE IN CUBICAL TYPE THEORY

EMIL HOLM GJØRUP AND BAS SPITTERS

INTRODUCTION

Interactive theorem provers can be a powerful tool for doing mathematics, but support from proving tedious lemmas can be improved. Decision procedures are very useful to automate such tasks. The congruence closure algorithm automates proofs that can be constructed using basic properties of equality: reflexivity, symmetry, transitivity and congruence. Congruence closure is a key algorithm in SMT solvers [2]. Coq contains the algorithm as a tactic to solve first order equational goals. In 2017 Selsam and de Moura presented an efficient procedure [5] to extend the algorithm to the Intensional Type Theory (ITT) of the Lean [3] proof assistant. To do so, they use UIP to find a workaround for the absence of the ideal congruence lemma (defined below). Instead, we observe that the ideal congruence lemma *is* well-typed and provable in Cubical Type Theory [1] (CTT) and this allows us to make a natural modification of their procedure and moreover eliminate the need for the UIP axiom. We present an implementation[1] in Cubical Agda [7].

IDEAL CONGRUENCE LEMMA

A key ingredient in the congruence closure algorithm is that functions respect equalities.

$$x = y \Rightarrow f\ x = f\ y$$

This, however, does not generalize well to ITT. The challenge arises when f is a *dependent* function $f : (a : A) \to B\ a$, so that $f\ x = f\ y$ does not type check. A solution is to use some notion of *heterogeneous equality*. Even more challenging, the lemma needed for the procedure is a generalization of congruence to handle equal functions as well.

(hcongr_ideal) $$f = g \wedge x = y \Rightarrow f\ x = g\ y$$

Where the functions can be of different type [5].

Heterogeneous equality. The definition of heterogeneous equality used by Selsam and de Moura [5] is the inductive type

$Heq : (A, B : \mathcal{U}) \to (a : A) \to (b : B) \to \mathcal{U}$

with the constructor $\mathit{hrefl} : (A : \mathcal{U}) \to (a : A) \to Heq\,A\,A\,a\,a$. It is almost the same as the definition of equality in ITT, with the exception of the added type B. But this choice of heterogeneous equality means that the `hcongr_ideal` is not provable. Instead Selsam and de Moura work around this by requiring the two functions f and g to be of same type and generate special case congruence lemmas for dependent functions taking n arguments. These special case lemmas can be proven using UIP.

Dependent path. In homotopical models of type theory a proof $p : a =_A b$ is interpreted as a path between two points a and b in a path space [6], which is a continuous map from the unit interval to a space $\mathbb{I} \to A$. This is taken literally in Cubical Type Theory [1] by adding the interval $\mathbb{I}$ together with a *path abstraction* $(\langle i \rangle\ t)$ to the type theory. Which is a special case of a more general path type namely a *dependent path*, where we are allowing the path space to depend on $\mathbb{I}$ as well $(i : \mathbb{I}) \to A\ i$. Using this as heterogeneous equality in Cubical Type Theory we are able to state and prove `hcongr_ideal`.

```
hcongr_ideal : ∀ {ℓ} {A : I → Type ℓ} {C : (i : I) → A i → Type ℓ}
                 {f : (a : A i0) → C i0 a} {g : (b : A i1) → C i1 b} {a : A i0} {b : A i1} →
                 (fg : PathP (λ i → (a : A i) → C i a) f g) → (ab : PathP A a b) →
                 PathP (λ i → C i (ab i)) (f a) (g b)
hcongr_ideal fg ab = λ i → fg i (ab i)
```

Spitters was supported by AFOSR project Homotopy Type Theory and Probabilistic Computation, 12595060.

[1]Implementation can be found at `https://github.com/limemloh/cubical-congruence`

CONGRUENCE CLOSURE PROCEDURE

The overall idea is to maintain an acyclic directed graph, which keeps track of equivalences between terms and have a another data structure for finding congruent terms, which can then be added as edges using the lemma from above. Each node in the graph represents a term either in the form of a function application of single argument $f\ a$ or some typing $a : A$, and which can be ensured by preprocessing. An edge represents an equality between two terms and the direction corresponds to the orientation of the equality. Every (weakly connected) component in the graph will have one *representative term*, where every

$$\begin{tikzcd}
& b & & & & \\
a \arrow[ur, "p"] & & c \arrow[ul, "q^{-1}"'] & d \arrow[l, "r^{-1}"'] & e \arrow[r, "s"] & f
\end{tikzcd}$$

FIGURE 1. An equivalence graph after adding $p : a = b$, $q : b = c$, $r : c = d$ and $s : e = f$

other term in the same component can iteratively follow outgoing edges and get to the representative term (similar to a `Union-find` data structure). We can then construct a proof of equality between two terms by collecting the edges connecting them to their shared representative, and fail in case they do not share representative.

The procedure by Selsam and de Moura [5] uses UIP to skip adding an edge, if the equivalence graph is already able to construct a proof of same equality type as the edge, resulting in every term only have up to one outgoing edge. But since we are without UIP; the goal can depend on a specific proof. The equivalence graph have to keep important proofs/edges. An example is the following problem:

$$(p,\ q : a = b) \to f\ a\ a =_{\langle i \rangle\ C\ (p\ i)\ (q\ i)} f\ b\ b$$

where $A : \mathcal{U}$, $C : A \to \mathcal{U}$, $a, b : A$ and $f : (a,\ a' : A) \to C\ a\ a'$. Here the goal depends on both p and q, meaning the graph need to contain both of them. We extend the equivalence graph to insert these edges, while still ensuring it to be acyclic and have representatives. The goal of the above example, depends twice on a proof of $a = b$ and unfortunately, it is difficult to know when to choose p or q when constructing the proof, since they are proofs of the same type and both appear in the goal. Our current approach is naive and constructs every possible combination of p and q, and then use the one which matches the goal. Meaning the procedure constructs up to n^2 proofs, if the goal is a dependent path depending on n paths and all n paths are in the same component of the equivalence graph.

Because we have the higher structure in CTT, the number of different proofs we can construct for $a = b$ while having both $p : a = b$ and $q : a = b$ in the equivalence graph potentially infinite, as we can cycle the proofs p and q^{-1} and form new proofs of $a = b$.

$$p \cdot q^{-1} \cdot p \cdot \ldots$$

Luckily, if the goal depends on a cycle, it would be part of the dependent path and we can extract this information, avoiding having to construct proofs using cycles.

IMPLEMENTATION

We have implemented the algorithm as a macro in Cubical Agda. The procedure relies heavily on map data structures and since Cubical Agda is still a young project, efficient data structures are still being developed. But even without efficient data structures our implementation is able to solve typical problems within seconds.

Remark. Unfortunately, stating `hcongr_ideal` in Homotopy Type Theory is hard compared to CTT. A solution using dependent paths has been proposed by Scoccola [4]. Similarly to the approach by Selsam and de Moura [5], this generates and proves special case lemmas for each dependent function on a meta level.

REFERENCES

[1] Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. "Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom." In: *FLAP* 4.10 (2017), pp. 3127–3170. URL: `http://collegepublications.co.uk/ifcolog/?00019`.

[2] Leonardo de Moura and Nikolaj Bjørner. "Z3: An Efficient SMT Solver." In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by C. R. Ramakrishnan and Jakob Rehof. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 337–340. ISBN: 978-3-540-78800-3.

[3] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. "The Lean Theorem Prover (System Description)." In: *Automated Deduction - CADE-25*. Ed. by Amy P. Felty and Aart Middeldorp. Cham: Springer International Publishing, 2015, pp. 378–388. ISBN: 978-3-319-21401-6.

[4] Luis Scoccola. *Congruence in Univalent Type Theory*. [Online; visited 15-may-2020]. June 2019. URL: `http://www.ii.uib.no/~bezem/abstracts/TYPES_2019_paper_38`.

[5] Daniel Selsam and Leonardo de Moura. "Congruence Closure in Intensional Type Theory." In: *CoRR* abs/1701.04391 (2017). arXiv: `1701.04391`. URL: `http://arxiv.org/abs/1701.04391`.

[6] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study: `https://homotopytypetheory.org/book`, 2013.

[7] Andrea Vezzosi, Anders Mörtberg, and Andreas Abel. "Cubical Agda: A Dependently Typed Programming Language with Univalence and Higher Inductive Types." In: *Proc. ACM Program. Lang.* 3.ICFP (July 2019). DOI: `10.1145/3341691`. URL: `https://doi.org/10.1145/3341691`.