A Tactic For Setoid Congruence

by

Kyle Ehrlich

A Major Qualifying Project

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Bachelor of Science

in Computer Science and Mathematical Sciences

by

Kyle Ehrlich

May 2021

APPROVED:

Daniel J. Dougherty, project adviser

Abstract

The congruence tactic built into the Coq proof management system allows for solving entailment of closed equalities with uninterpreted function symbols. In this project we build a congruence tactic that works with multiple relations. My theoretical contribution is to describe a translation from entailments in a generalized form of congruence into the traditional form that existing algorithms can solve. The tactic makes use of this transformation, as well as an implementation of congruence closure, to solve Coq goals involving multiple relations.

Contents

Chapter 1

Introduction

1.1 Domain

Congruence relations are important in mathematics. Congruence closure is an important algorithm for working with congruence relations, it allows us to surprisingly efficiently answer the question: "does one finite set of equations imply a specific equation?" While this problem is answerable without considering congruence relations directly, as in the approach described by Ackermann [Ack54], considering congruence relations allow for efficient algorithms, such as the one described by Nelson and Oppen, [NO80] for the question as well as more broad applications.

One such application lives is within Coq the interactive theorem prover Coq. In Coq proofs are built up out of tactics. One such tactic is the congruence tactic which is powered by an extended version of the congruence closure algorithm devised by Corbineau [Cor01] in 2001 that allows for the generation of proofs. This tactic allows the user to automatically solve goals using properties of equality.

1.2 Objective

My objective was to generalize this tactic to be able to work with relations beyond equality, such as equivalence in modular arithmetic. Furthermore we want our tactic to be able to able work with several such congruence relations simultaneously.

1.3 Contribution

My theoretical contribution is a translation from a more general form of the entailment problem that allows for multiple relations into the traditional entailment problem solved by congruence closure.

My implementation contribution is a tactic implementing this translation. It is of note that the built-in congruence tactic is implemented as a Coq Plugin, whereas our tactic is implemented in the new Ltac2 meta-programing language.

1.4 Related Work

The first work cited as showing the decidability of entailment of finite uninterpreted equalities is [Ack54]. Kozen [Koz77] used a graph based approach to show that the problem is P-complete. Nelson and Oppen [NO80] show a quadratic procedure based off congruence closure used . They then apply this to the the more specific case of the theory of lists built with cons and nil. Downey, Ravi and Tarjan [DST80] show a faster method for the same question, viewed though the lens of the common sub expression problem. Shostak [Sho78] [Sho82] [CLS96] describes yet another approach and gives a clear proof of the correspondence between congruences and uninterpreted equalities. Nieuwenhuis [NO05] describes modified congruence algorithms to produce proofs. Corbineau [Cor01] [Cor06] describes the tagging system used to

produce proofs by the congruence tactic, and describes the theoretical extensions behind the congruence tactic's extended support for constructor theory.

Chapter 2

Background

In this chapter I will discuss the start of entailment of ground equalities. I will then discuss how this problem can be re-framed in terms of congruence relations, and how this enables the creation of practical algorithms. Lastly I will explain how congruence closure powers `congruence` tactic and how the limitations of the congruence tactic motivates my tactic and the necessary generalizations.

2.1 Entailment Of Equalities

An entailment of equalities question is structured as follows: "for any interpretation of the constant symbols and functions such that some finite set of ground equalities hold, must another hold?"

This entailment features many properties we are used to working with. Namely for any terms a, b, and c:

1. any set of equations entails $a = a$, by the reflexive property of equality
2. $a = b$ entails $b = a$ by the symmetric property of equality
3. $a = b$ and $b = c$ together entail $a = c$ by the transitive property of equality

4. $a = b$ entails $T[a] = T[b]$ by the substitution property of equality

Example 1. The two equations $f(f(f(f(f(x))))) = x$ and $f(f(f(x))) = x$ together entail $f(x) = x$.

We can see this because for any interpretations of f and x, say $f_{\mathcal{M}}$ and $x_{\mathcal{M}}$ such that $f_{\mathcal{M}}(f_{\mathcal{M}}(f_{\mathcal{M}}(f_{\mathcal{M}}(f_{\mathcal{M}}(x_{\mathcal{M}}))))) = x_{\mathcal{M}}$ and $f_{\mathcal{M}}(f_{\mathcal{M}}(f_{\mathcal{M}}(x_{\mathcal{M}}))) = x_{\mathcal{M}}$. By substituting the first equality into the second we conclude that $f_{\mathcal{M}}(f_{\mathcal{M}}(x_{\mathcal{M}})) = x_{\mathcal{M}}$. By substituting that new equality into the last we can conclude that $f_{\mathcal{M}}(x_{\mathcal{M}}) = x_{\mathcal{M}}$, completing the entailment.

Example 2. The equation $f(x, y) = f(y, x)$ does not entail $f(x, f(x, y)) = f(f(y, x), x)$.

We see this by constructing an interpretation such that the hypotheses holds but the conclusion does not. An example of an interpretation that works is $x_{\mathcal{M}} \mapsto 2$ $y_{\mathcal{M}} \mapsto 4$ where we interpret $f_{\mathcal{M}}(a, b) \mapsto a^b$ with your interpretation's domain being the natural numbers. It should be clear that $f_{\mathcal{M}}(x_{\mathcal{M}}, y_{\mathcal{M}}) = 2^4 = 16 = 4^2 = f_{\mathcal{M}}(y_{\mathcal{M}}, x_{\mathcal{M}})$, while $f_{\mathcal{M}}(x_{\mathcal{M}}, f_{\mathcal{M}}(x_{\mathcal{M}}, y_{\mathcal{M}})) = 2^{2^4} \neq (4^2)^2 = f_{\mathcal{M}}(f_{\mathcal{M}}(y_{\mathcal{M}}, x_{\mathcal{M}}), x_{\mathcal{M}})$.

In 1954 Ackermann [Ack54], showed that you can limit the size of any counterexample. Specifically given a counterexample, if you restrict the domain to only values that arise from terms (including subterms) that occur in either the hypothesis or conclusion, you still have a counterexample.

By limiting the size of a counterexample, it is possible to enumerate all possible counter examples. This gives a decision procedure for entailment, but not a practical one. Alone, the small example has 6 unique subterms giving rise to $6 * 6 * 6^{6*6} \approx 3.71 * 10^{29}$ possible counter examples if we take the naive approach.

2.2 Congruence Relations

It is possible to view entailment of uninterpreted equalities as quantifying over the interpretation of the equality symbol rather than quantifying over the interpretation of the term. In this case we ask "if for every **congruence relation** such that one finite set of equations holds, must another hold?" where a **congruence relation** is any equivalence relation where each function symbol acts properly on the equivalence classes.

This is captured by the congruence axiom:[1]

$$\frac{s_1 \sim t_1 \qquad \dots \qquad s_n \sim t_n}{f(s_1 \dots s_n) \sim f(t_1 \dots t_n)} \text{ CONG}$$

Nelson and Oppen [NO80] show that these two perspectives are equivalent, and provide an efficient algorithm for computing inference in congruence relations.

We can work though this for the same examples as above:

Example 3. The two equations $f(f(f(f(f(x))))) = x$ and $f(f(f(x))) = x$ together entail $f(x) = x$.

Given some $\sim$ such that $f(f(f(f(f(x))))) \sim x$ and $f(f(f(x))) \sim x$ we can construct the following inference tree:

$$\dfrac{\dfrac{\dfrac{f^5(x) \sim x}{x \sim f^5(x)} \text{ SYM} \quad \dfrac{\dfrac{f^3(x) \sim x}{f^4(x) \sim f(x)} \text{ CONG}}{f^5(x) \sim f(f(x))} \text{ CONG}}{\dfrac{x \sim f(f(x))}{f(x) \sim f(f(f(x)))} \text{ CONG} \qquad f(f(f(x))) \sim x} \text{ TRANS}}{f(x) \sim x} \text{ TRANS}$$

Example 4. The equation $f(x, y) = f(y, x)$ does not entail $f(x, f(x, y)) = f(f(y, x), x)$.

We construct a congruence relation that forms a counterexample:

One such congruence relation that works is $t_1 \sim t_2$ iff the depth of the left-most constant symbol is identical in both.

[1] For a rigorous description of this notation, see Chapter 3

2.3 Coq

To quote the Coq landing page:

> Coq is a formal proof management system. It provides a formal language to write mathematical definitions, executable algorithms and theorems together with an environment for semi-interactive development of machine-checked proofs. [coq]

Well-known applications of Coq includes the verified C compiler and verification of the four color theorem.

It's important to note that Coq is not a push-button theorem prover, rather proofs in Coq are built by invoking a series of tactics. These can be push-buttons for specific problems.

2.3.1 The congruence tactic

The congruence tactic was designed by Corbineau in 2001, a few years later he extended it to include additional support for constructor theory, which he gave theoretical backing to in a 2006 paper [Cor06].

The tactic takes advantage of the fact that each of the axioms that define a congruence relations, namely reflexivity, symmetry, transitivity, and congruence are all provable in Coq. This means that in any Coq proof context, terms being probably equal is a congruence relation. As such, any inferences derived by congruence closure can be proven in any Coq context.

Example 5. Following the structure from earlier, the congruence tactic can instantly complete the proof from the following state:

```
x: nat
```

```
f: nat -> nat
H1: f (f (f (f (f x)))) = x
H2: f (f (f x)) = x
----------------------------
(1/1)
f x = x
```

For comparison here is the same proof accomplished using the rewrite tactic.

```
rewrite ← H1 at 1.
rewrite -> H2.
exact H2.
```

Proofs of this sort can quickly become awkward as terms grow large.

2.4 Limitations Of Congruence

The congruence tactic works exclusively on the Logic.eq relations built into Coq. Logic.eq is defined as follows:

```
Inductive eq (A:Type) (x:A) : A -> Prop :=
  eq_refl : eq x x
```

The important thing about this definition is that in the case of closed terms it only holds for terms that can be reduced to identical terms using Coq's computation method.

This is often stronger than you want. Take for example the rational numbers. Coq represents these as simply a pair of a signed integer and a positive integer, representing the numerator and denominator respectively. The effect of this is that you can have two representations of the same rational number that are not Logic.eq,

for example $(2, 4)$ and $(1, 2)$ both encode $\frac{1}{2}$ but are distinct fully reduced terms. Instead to talk about rational equality you need to use the QEq relation, which is defined to capture rational equivalence. It's important to note that this equivalence relation of pairs is by no means unique to Coq, the difference is that in most contexts you would immediately quotient

Notice that for some function symbols, QEq provably still obeys the congruence axiom, for example

$$\frac{a \approx_{\text{Qeq}} b \qquad c \approx_{\text{Qeq}} d}{\text{plus}(a, c) \approx_{\text{Qeq}} \text{plus}(b, d)} \text{CONG}$$

With this in mind my goal is to build a congruence tactic that can work with relations beyond Logic.eq.

2.5 Complications

Unfortunately there are complications that prevent us from directly applying existing congruence closure algorithms to this scenario.

2.5.1 Partial congruences

There are some function symbols where the congruence axiom is false for some relations. For example, the Qnum function that extracts the numerator of a rational, can easily map two rationals that are Qeq onto distinct integers.

2.5.2 Multiple relations

Often times in Coq states you can be considering multiple relations at once, this becomes an issue when they are for the same sort.

For example you might want to consider Qeq and Logic.eq at the same time when using the QRed function which maps rationals into simplest form, for example QRed (2 / 4)= 1 / 2. This function takes any two rationals that are Qeq and returns rations that are Logic.eq, we can capture this using a congruence-like axiom.

$$\frac{a \sim_{\text{Qeq}} b}{\text{Qred}(a) \sim_{\text{eq}} \text{Qred}(b)}$$

In geneal form these congruence axiom take the form

$$\frac{s_1 \sim_{r_1} t_1 \qquad \dots \qquad s_n \sim_{r_n} t_n}{f(s_1 \dots s_n) \sim_{r'} f(t_1 \dots t_n)} \text{CONG}$$

But this extension isn't compatible with existing congruence closure algorithms. To see why, it is important to note that most congruence closure algorithms reply on the fact that you only need to consider terms that are subterms of your hypotheses or conclusion, but this is not true in the general multi-relation context.

Example 6. Say we have the two congruence inferences:

$$\frac{a \sim_2 b}{f(a) \sim_1 f(b)}$$

$$\frac{a \sim_3 b}{f(a) \sim_1 f(b)}$$

The two equations $x \sim_2 y$ and $y \sim_3 z$ together entail $f(x) \sim_1 f(z)$, but any proof of this fact must involve the term $f(y)$ which is not a subterm of the conclusion or the goal.

An common case where this sort of parody can come up is with subrelations. If $\sim_2$ is a subrelation of $\sim_3$, than the second axiom in the example naturally entails the first. As such it will prove helpful to include subrelation inferences in our theory to help resolve these patterns.

2.6 Problem

My theoretical contribution is to describe a procedure for removing such patterns, and once those patterns are removed describe a procedure for mapping entailment questions in the multi-relation theory into traditional single relation congruence.

From an implementation perspective, the information I need is surfaced by Coq's typeclasses. Specifically of interest are the 'Proper' and the 'subrelation' typeclasses. These typeclasses are already used by Coq's support for setoid rewriting, so it is not unreasonable to expect users to have the information already registered. [Soz]

Chapter 3

Theory

3.1 Preliminaries

Definition 7 (Horn Sentence)**.** *A Horn sentence is a sentence of the form*

$$\forall x_1 \dots x_m, \bigwedge_i P_i(t_{i,1}, \dots, t_{i,k}) \to Q(u_1 \dots u_k)$$

Where each $t_{i,j}$ and each u_j is a term over $x_1 \dots x_m$.

Definition 8 (Sorted Set)**.** *An S-sorted set is a set X with an associated function from X to S. We treat X as referring to the set and $\mathrm{sort}_X(t)$ for any $t \in X$ as referring to the label of t.*

We say that two sorted sets are subsets of each other if they are subsets considered as sets and all of their common elements have the same label.

An equivalent way of viewing sorted sets is to view a S sorted set $X = \{X_i | i \in \mathcal{R}\}$ as a collection of disjoint sets indexed by $\mathcal{R}$. In this notation we say that $x \in X$ if $\exists i, x \in X_i$.

These two notions are equivalent by the construction $X_i = \{x \mid x \in X, \mathrm{sort}_X(x) =$

$s\}$ and $\text{sort}_X(x) = i$ iff $x \in X_i$.

Definition 9 (Sorted Set Product). *Using the definitions* $\text{sort}_{A\times B}((a,b)) = (\text{sort}_A(a), \text{sort}_B(b))$ *we can lift the Cartesian product to sorted sets.*

Likewise for $\text{sort}_{A^*}((a_1,\ldots,a_n)) = (\text{sort}_A(a_1),\ldots,\text{sort}_A(a_n))$

Definition 10 (Term Signature). *A term signature is a pair* $(\mathcal{S},\Sigma)$ *where* Σ *is a* $\mathcal{S}^* \times \mathcal{S}$*-sorted set.*

For any $f \in \Sigma$ *with* $\text{sort}_\Sigma(f) = ((s_1 \ldots s_n), s)$ *we say that* s *is the return sort of* f *and that* n *is the arity of* f.

The $\mathcal{S}$*-sorted set of well-sorted terms* $\mathcal{T}(\mathcal{S},\Sigma)$ *constructed by finite applications of each function symbol.*

Definition 11 (Well-Sorted Relation). *We say that a relation* $\sim$ *on an* S*-sorted set* T *is* ***well-sorted****if and only if for any* $x, y \in T$ *such that* $x \sim y$, $\text{sort}_T(x) = \text{sort}_T(y)$

Going forward all named relations will be assumed to be well-sorted.

Definition 12 (Equivalence Relation). *An equivalence relation is a relation* $\sim$ *that is transitive, reflexive and symmetric.*

Definition 13 (Uniform Congruence Relation). *Let* $(\mathcal{S},\Sigma)$ *be a term signature. A* ***uniform congruence relation*** *is a well-sorted equivalence relation* $\sim$ *on* $\mathcal{T}(\mathcal{S},\Sigma)$ *such that for all* $f \in \Sigma$, $s_1 \ldots s_n, t_1 \ldots t_n \in \mathcal{T}(\mathcal{S},\Sigma)$ *such that* $f(s_1 \ldots s_n)$ *and* $f(t_1 \ldots t_n)$ *are well-sorted terms, if for all* i, $s_i \sim t_i$ *then* $f(s_1 \ldots s_n) \sim f(t_1 \ldots t_n)$.

This gives rise to inference notation for congruence that we introduced in 2.5.2.

$$\frac{s_1 \sim t_1 \qquad \ldots \qquad s_n \sim t_n}{f(s_1 \ldots s_n) \sim f(t_1 \ldots t_n)}\ \text{CONG}$$

3.2 Formalizing multi-relation congruence

First we need to define what our models are, specifically we are going to looking at families of relations, where each relation operates on a single sort. This is different from uniform congruence, which considers a single well-sorted relation.

Definition 14 (Relation Family)**.** *Let be* $(\mathcal{S}, \Sigma)$ *a term signature and* $\mathcal{R}$ *be a* $\mathcal{S}$*-sorted set of relation symbols where the sort of of a relation symbol is the sort of terms related. A relation family is a set of triples of the form* (i, x, y) *where* i *is some member of the index set, and* x, y *are closed terms such that* i*,* x *and* y *all have the same sort.*

We interpret a relation family $\sim$ *as a family of relations* $\sim_i$ *over the term model where* $x \sim_i y$ *if and only if* $(i, x, y) \in \sim$*.*

We say it is an ***equivalence relation family*** *for each* $i \in \mathcal{R}$*,* $\sim_i$ *forms an equivalence relation on* $\mathcal{T}(\mathcal{S}, \Sigma)_{\mathrm{sort}_{\mathcal{R}}(i)}$*.*

For the remainder of this section unless otherwise noted we will consider a fixed $\mathcal{S}$, Σ and $\mathcal{R}$.

Definition 15 (T-Relation Family)**.** *Given* T*, some class of well-sorted sentences with relation symbols in* $\mathcal{R}$ *and terms in* U*, we can say that a relation family is a* T*-relation family if it satisfies the assertions of* T*.*

Example 16. Over the term signature $(\{s_0\}, \{\text{true}, \text{false}\})$ with index set $\mathcal{R} = \{1, 2\}$ where each symbol has arity 0, if we select T to be comprised of the singe sentence $\forall xy, (x \sim_1 y \wedge y \sim_1 x) \rightarrow x \sim_2 y$.

$\{(1, \text{false}, \text{true}), (1, \text{true}, \text{true})\}$ is not a T-relation family since it doesn't satisfy the assertions of the second sentence in T.

$\{(1, \text{false}, \text{true}), (1, \text{true}, \text{true}), (2, \text{true}, \text{true}), (2, \text{false}, \text{false})\}$ is a T-relation family.

Lemma 17. *When T is a set of Horn sentences, the intersection of T-relation families is itself a T-relation family.*

Definition 18. *Therefore we can conclude that the for some relation family $\diamond$ the intersection of all T-relation families that include $\diamond$ is a T-relation family, we call this the T-relation family generated by $\diamond$.*

Next we build a notion of subrelations from an ordering on the index set.

Definition 19. *Given some partial ordering on $\mathcal{R}$ set $\leq$, we say that a relation family $\sim$ obeys $\leq$ if and only if for every $r \leq r'$ it satisfies:*

$$\frac{s \sim_r t}{s \sim_{r'} t} \text{SUBR}_{r \leq r'}$$

Next we need to formalize our list of axioms, to do this we define a congruence signature which is basically a set form of congruence axioms.

Definition 20 (Congruence Signature)**.** *Given some term signature $(\mathcal{S}, \Sigma)$ and a $\mathcal{S}$-sorted set $\mathcal{R}$. A congruence signature $\mathcal{C}$ is a set of tuples of the form $(f, (s_1 \dots s_n), s)$ where $f \in \Sigma$, $s_1 \dots s_n, s \in \mathcal{R}$ such that $\text{sort}_\Sigma(f) = ((\text{sort}_\mathcal{R}(s_1) \dots \text{sort}_\mathcal{R}(s_n)), \text{sort}_\mathcal{R}(s))$.*

An equivalence relation family with relation symbols $\mathcal{R}$ over $\mathcal{T}(\mathcal{S}, \Sigma)$, $\sim$ is a ***$\mathcal{C}$-congruence*** *if and only if for each $(f, (r_1, \dots, r_n), r) \in \mathcal{C}$, for each $s_1, \dots, s_n, t_1, \dots t_n$ such that $f(s_1 \dots s_n), f(t_1 \dots t_n) \in \mathcal{T}(\mathcal{S}, \Sigma)$ if for each i, $s_i \sim_{r_i} t_i$ then $f(s_1 \dots s_n) \sim_r f(t_1 \dots t_n)$*

As this is a Horn sentence we can notate it:

$$\frac{s_1 \sim_{r_1} t_1 \quad \dots \quad s_n \sim_{r_n} t_n}{f(s_1 \dots s_n) \sim_r f(t_1 \dots t_n)} \mathcal{C}$$

Example 21. In our example the used $\mathcal{C}$ is:

$$\mathcal{C}_{\mathbb{Q}} = \{(\text{plus}, (\text{Qeq}, \text{Qeq}), \text{Qeq}),$$
$$(\text{plus}, (\text{eq}, \text{eq}), \text{eq})$$
$$(\text{Qred}, (\text{Qeq}), \text{eq})\}$$

We notice however that this notion of a congruence has implied inferences that are not included in the set.

Example 22. Consider the signature $(\mathcal{S}, \Sigma) = (\{k\}, \{f\})$ where $\text{sort}_\Sigma(f) = ((k, k), k)$ with $\mathcal{R} = \{a, b, c\}$, and congruence signature $\mathcal{C} = \{(f, (a, a), c), (f, (b, b), c)\}$.

Notice that any $\mathcal{C}$-congruence $\sim$ will also obey the inference associated with $(f, (a, b), c)$:

$$\frac{s_1 \sim_a t_1 \qquad s_2 \sim_b t_2}{f(s_1, s_2) \sim_c f(t_1, t_2)}$$

This is because given any $s_1, t_1, s_2, t_2 \in \mathcal{T}(\mathcal{S}, \Sigma)_k$ such that $s_1 \sim_a t_1$ and $s_2 \sim_b t_2$, we know the following:

1. $f(s_1, s_2) \sim_c f(t_1, s_2)$ because $s_2 \sim_a s_2$ by reflexivity and $(f, (a, a), c) \in \mathcal{C}$
2. $f(t_1, s_2) \sim_c f(t_1, t_2)$ because $t_1 \sim_b t_1$ by reflexivity and $(f, (b, b), c) \in \mathcal{C}$

We combine these with transitivity to conclude that $f(s_1, s_2) \sim_c f(t_1, t_2)$.

This proof can be viewed as a tree:

$$\dfrac{\dfrac{s_1 \sim_a t_1 \qquad \dfrac{}{s_2 \sim_a s_2}\,\text{REFL}}{f(s_1, s_2) \sim_c f(t_1, s_2)}\,\mathcal{C} \qquad \dfrac{\dfrac{}{t_1 \sim_b t_1}\,\text{REFL} \qquad s_2 \sim_b t_2}{f(t_1, s_2) \sim_c f(t_1, t_2)}\,\mathcal{C}}{f(s_1, s_2) \sim_c f(t_1, t_2)}\,\text{TRANS}$$

This works because it suffices to apply the congruence axioms by only changing one argument at a time, recombining using transitivity. To address this we introduce a simplified notion of a congruence signature that considers only one argument per entry.

We follow a similar construction as normal congruence congruence signature, in essence these capture axioms like those captured by congruence signatures, except that all but one of the pairs of corresponding arguments are instantiated at the same terms.

Definition 23 (Linear congruence signature)**.** *A linear congruence signature* $\mathcal{M}$ *is a set of tuples of the form* $(f, k, r, r') \in \Sigma \times \mathbb{Z}^+ \times \mathcal{R} \times \mathcal{R}$ *where the sort of* r *matches the sort of the* k*th argument of* f *and the sort of* r' *matches the return sort of* f*.*

An equivalence relation family $\sim$ *is a* $\mathcal{M}$*-congruence linear congruence signature* $\mathcal{M}$ *if for each* $(f, k, r, r') \in \mathcal{M}$ *it satisfies the following axiom:*

$$\frac{x \sim_r y}{f(s_1, \dots s_{k-1}, x, s_{k+1}, \dots s_n) \sim_{r'} f(s_1, \dots s_{k-1}, y, s_{k+1}, \dots s_n)} \text{CONGR}_{\mathcal{M}}$$

Definition 24 (Induced Linear congruence signature)**.** *Given some congruence signature* $\mathcal{C}$ *we can define an induced linear congruence signature as follows*

$$\mathcal{M}(\mathcal{C}) = \{(f, k, r_k, r) \mid (f, (r_1, \dots r_k, \dots, r_n), r) \in \mathcal{C}\}$$

Example 25. In our example the induced $\mathcal{M}(\mathcal{C}_{\mathbb{Q}})$ is:

$$\begin{aligned}\mathcal{M}(\mathcal{C}) = \{&(\text{plus}, 1, \text{Qeq}, \text{Qeq}),\\ &(\text{plus}, 2, \text{Qeq}, \text{Qeq}),\\ &(\text{plus}, 1, \text{eq}, \text{eq}),\\ &(\text{plus}, 2, \text{eq}, \text{eq}),\\ &(\text{Qred}, 1, \text{Qeq}, \text{eq})\}\end{aligned}$$

Example 26. Continuing from Example 22 the induced $\mathcal{M}(\mathcal{C})$ is:

$$\begin{aligned}\mathcal{M}(\mathcal{C}) = \{&(f, 1, a, a),\\ &(f, 2, a, a),\\ &(f, 1, b, a),\\ &(f, 2, b, a), \}\end{aligned}$$

Notice here that all induced single argument congruences are now covered.

Next we show formal equivalence between these two perspectives

Lemma 27. *An equivalence relation family $\sim$ with relation symbols $\mathcal{R}$ over $\mathcal{T}(\mathcal{S}, \Sigma)$ is a $\mathcal{C}$-congruence if and only if it is a $\mathcal{M}(\mathcal{C})$-congruence.*

Proof. Given that $\sim$ is a $\mathcal{C}$-congruence we need to show that it is also a $\mathcal{M}(\mathcal{C})$-congruence. To do this we must show that for all $(f, k, r, r') \in \mathcal{M}(\mathcal{C})$ if $x \sim_r y$ then $f(s_1, \ldots s_{k-1}, x, s_{k+1}, \ldots s_n) \sim_{r'} f(s_1, \ldots s_{k-1}, y, s_{k+1}, \ldots s_n)$. By definition since $(f, k, r, r') \in \mathcal{M}(\mathcal{C})$ we know that for some $r_1 \ldots r_n$, $(f, (r_1, \ldots r_{k-1}, r, r_{k+1}, \ldots r_n), r') \in \mathcal{C}$. Since $\sim$ is a $\mathcal{C}$-congruence by reflexivity we know that for each i, $s_i \sim_{r_i} s_i$, so so by

congruence we know that $f(s_1, \ldots s_{k-1}, x, s_{k+1}, \ldots s_n) \sim_{r'} f(s_1, \ldots s_{k-1}, y, s_{k+1}, \ldots s_n)$.

For the other direction given that $\sim$ is a $\mathcal{M}(\mathcal{C})$-congruence we need to show that it is also a $\mathcal{C}$-congruence. To do this we must show that for all $(f, (r_1 \ldots r_n), r') \in \mathcal{C}$, if for all k, $s_k \sim_{r_k} t_k$ then $f(s_1 \ldots s_n) \sim_{r'} f(t_1 \ldots s_n)$. To do this we let $x_i = f(t_1 \ldots t_i, s_{i+1} \ldots s_n)$ where $x_0 = f(s_1 \ldots s_n)$ and $x_{n+1} = f(t_1 \ldots t_n)$. For a given i we observe that by definition $(f, i, r_i) \in \mathcal{M}(\mathcal{C})$ since $s_i \sim_{r_i} t_i$ this tells us by linear congruence that $x_{i-1} = f(t_1 \ldots t_{i-1}, s_i, s_{i+1} \ldots s_n) \sim_{r'} f(t_1 \ldots t_{i-1}, t_i, s_{i+1} \ldots s_n) = x_i$. We conclude that by transitivity $f(s_1 \ldots s_n) = x_0 \sim_{r'} x_1 \ldots x_n \sim_{r'} x_{n+1} = f(t_1 \ldots t_n)$.

□

Corollary 28. *For any relation family $\diamond$.*

The $\mathcal{C}$,$\leq$-congruence generated by $\diamond$ is identical to the $\mathcal{M}(\mathcal{C})$,$\leq$-congruence generated by $\diamond$.

3.3 Reducing to single relation congruence

Now we need to set about reducing multi-relation congruence to a problem in single relation congruence. The general approach we are going to take is to model multi-relation congruence in single relation congruence by introducing new terms that represent an original term paired with a relation. In order to be able to do this, we need to have a unique choice of relations to consider subterms under, so we define conditions on a linear congruence signaturethat make this the case:

If a linear congruence signature is total and unique in the second-to-last argument we say it is a congruence signature function. Formally:

Definition 29 (Congruence Signature Function)**.** *A linear congruence signature $\mathcal{M}$ is a* congruence signature function *if for each $f \in \Sigma, k \in \mathbb{Z}^+$ and $r \in \mathcal{R}$ such that f*

has no more than k arguments and the return sort of r matches the return sort of with $\mathrm{snd}(\mathrm{sort}_{\mathrm{ARG}(\Sigma)}((f,k))) = \mathrm{sort}_{\mathcal{R}}(r)$ *there exists exactly one r_k with $(f,k,r_k,r) \in \mathcal{M}$.*

We denote this r_k as $\Theta(f,k,r)$.

It is of note that a congruence signature function is just a special case of a linear congruence signature.

We observe that we can easily construct a congruence signature from a congruence signature as follows, if Θ is a congruence signature function than we can construct a congruence signature

$$\mathcal{C} = \{(f, (\Theta(f,1,r) \ldots \Theta(f,n,r)), r)\}$$

such that $\mathcal{M}(\mathcal{C}) = \Theta$.

Example 30. For our running example, we can read the following values for $\mathcal{C}_{\mathbb{Q}}$ directly out of of our $\mathcal{M}(\mathcal{C}_{\mathbb{Q}})$

$$\mathcal{C}_{\mathbb{Q}}(\mathrm{plus}, 1, \mathrm{Qeq}) = \mathrm{Qeq}$$
$$\mathcal{C}_{\mathbb{Q}}(\mathrm{plus}, 2, \mathrm{Qeq}) = \mathrm{Qeq}$$
$$\mathcal{C}_{\mathbb{Q}}(\mathrm{plus}, 1, \mathrm{eq}) = \mathrm{eq}$$
$$\mathcal{C}_{\mathbb{Q}}(\mathrm{plus}, 2, \mathrm{eq}) = \mathrm{eq}$$
$$\mathcal{C}_{\mathbb{Q}}(\mathrm{Qred}, 1, \mathrm{eq}) = \mathrm{Qeq}$$

Note that this is incomplete, it is missing a value at $\Theta(\mathrm{Qred}, 1, \mathrm{Qeq})$.

If we consider the subrelation structure however it would be perfectly acceptable

to add:

$$\Theta(\text{Qred}, 1, \text{Qeq}) = \text{Qeq}$$

Lets look at what a case where you can't build a morphism function looks like a simple example is $\mathcal{M} = \{(f, 1, a, c), (f, 1, b, c)\}$ leaves us with an ambiguous value for $\Theta(f, 1, c)$ How could we get this unstuck? Noted in Section 2.5.2, this sort of pattern often comes up with subrelation structures so say that we had $a \leq b$. In this case the sentence associated with $(f, 1, a, c)$ is redundant, because we can accomplish the same inference like so:

$$\frac{\dfrac{x \approx_a y}{x \approx_a y}\, a \leq b}{f(x) \approx_c f(y)}\mathcal{M}$$

This motivates the definition of resolvable, which requires that you have a largest relation symbol in all conflicts.

Definition 31 (Resolvable)**.** *A linear congruence signature $\mathcal{M}$ with $\leq$ is resolvable iff for every $f \in \Sigma, k \in \mathbb{Z}^+, r' \in \mathcal{R}$ where f has at least k arguments, and the sort of r' matches the return sort of f, the set $\{r | \exists a, (f, k, r, a) \wedge a \leq r'\}$ has a maximum. In such a case we say that $\Theta_{\mathcal{M},\leq}(f, k, r')$ is a function that computes such such a the maximum of each.*

Note that this always holds if $\leq$ is totally ordered on the relations of each sort. In our Coq context this will always happen when considering at most one relation per sort in addition to eq.

Lemma 32. *If a linear congruence signature $\mathcal{M}$ with an ordering $\leq$ is resolvable, than any relation family $\sim$ is a $\mathcal{M}$,$\leq$-congruence iff it is a $\Theta_{\mathcal{M},\leq}$,$\leq$-congruence.*

Proof. Given that $\sim$ is a $\mathcal{M},\leq$-congruence we need to show that it is a $\Theta_{\mathcal{M},\leq},\leq$-congruence. To do that we need to show that given some f, k, r' that satisfy the bounds given in Definition 31, we need to satisfy the following Horn sentence:

$$\frac{x \sim_{\Theta_{\mathcal{M},\leq}(f,k,r')} y}{f(s_1, \dots s_{k-1}, x, s_{k+1}, \dots s_n) \sim_{r'} f(s_1, \dots s_{k-1}, y, s_{k+1}, \dots s_n)}$$

By definition there must exist some $a \in \mathcal{R}$ such that $a \leq r'$ and $(f, k, \Theta_{\mathcal{M},\leq}(f, k, r'), a) \in \mathcal{M}$. Therefore given some instantiation of the Horn sentence we know that $f(\dots, x, \dots) \sim_a f(\dots, y, \dots)$ which since $a \leq r'$ tells us that $f(\dots, x, \dots) \sim_{r'} f(\dots, y, \dots)$.

Captured as a proof tree this can be formulated as:

$$\dfrac{\dfrac{x \sim_{\Theta_{\mathcal{M},\leq}(f,k,r')} y}{f(s_1, \dots s_{k-1}, x, s_{k+1}, \dots s_n) \sim_a f(s_1, \dots s_{k-1}, y, s_{k+1}, \dots s_n)}\,\mathcal{M}}{f(s_1, \dots s_{k-1}, x, s_{k+1}, \dots s_n) \sim_{r'} f(s_1, \dots s_{k-1}, y, s_{k+1}, \dots s_n)}\,\text{a} \leq \text{r'}$$

In the opposite direction, given that $\sim$ is a $\Theta_{\mathcal{M},\leq},\leq$-congruence we want to show that it is a $\mathcal{M},\leq$-congruence given some $(f, k, r, r') \in \mathcal{M}$ we need to satisfy the following inference:

$$\frac{x \sim_r y}{f(s_1, \dots s_{k-1}, x, s_{k+1}, \dots s_n) \sim_{r'} f(s_1, \dots s_{k-1}, y, s_{k+1}, \dots s_n)}$$

By picking $a = r'$ then $a \leq r'$ so we know that $r \in \{r | \exists a, (f, k, r, a) \wedge a \leq r'\}$. Since $\Theta_{\mathcal{M},\leq}(f, k, r')$ is the maximal element of that set by definition $r \leq \Theta_{\mathcal{M},\leq}(f, k, r')$ therefore we can construct the inference as follows:

$$\dfrac{\dfrac{x \sim_r y}{x \sim_{\Theta_{\mathcal{M},\leq}(f,k,r')} y}\leq}{f(s_1, \dots s_{k-1}, x, s_{k+1}, \dots s_n) \sim_{r'} f(s_1, \dots s_{k-1}, y, s_{k+1}, \dots s_n)}\,\Theta_{\mathcal{M},\leq}$$

□

Corollary 33. *If a linear congruence signature, $\mathcal{M}$ with an ordering $\leq$ is resolvable, for any relation family $\diamond$, the $\mathcal{M},\leq$-congruence generated by $\diamond$ is also the $\Theta_{\mathcal{M},\leq},\leq$-congruence generated by $\diamond$*

Lemma 34. *Let Θ be a congruence signature function such that for all a, b if $a \leq b$ then $\Theta(f, k, a) \leq \Theta(f, k, b)$ and $\bowtie$ be some relation family that respects $\leq$. The Θ-congruence generated by $\bowtie$ respects $\leq$.*

Proof. Let $\sim$ be the Θ-congruence generated by $\bowtie$. We define a new relation family $\diamond$ as follows: $x \diamond_r y$ if and only if for all $r' \geq r$, $x \sim_{r'} y$. By definition, $\diamond$ respects $\leq$. Additionally, $\diamond$ is an equivalence relation family.

Lastly to see that $\diamond$ is a Θ-congruence we want to show that:

$$\frac{s_k \diamond_{\Theta(f,k,r)} t_k}{f(\overline{s}_{1\ldots k-1}, s_k, \overline{s}_{k+1\ldots n}) \diamond_r f((\overline{s}_{1\ldots k-1}, t_k, \overline{s}_{k+1\ldots n})} \text{CONGR}_{\mathcal{M}}$$

Because we are given that for all $r' \geq \Theta(f, k, r)$, $s_k \sim_{\Theta(f,k,r')} t_k$ and that Θ is non-decreasing we know that for all $r' \geq r$, $s_k \sim_{\Theta(f,k,r')} t_k$. By congruence we can conclude that for all $r' \geq r$, $f(\overline{s}_{1\ldots k-1}, s_k, \overline{s}_{k+1\ldots n}) \sim_{r'} f((\overline{s}_{1\ldots k-1}, t_k, \overline{s}_{k+1\ldots n})$.

Therefore $\diamond$ is a Θ-congruence that includes $\bowtie$, so by definition it must include $\sim$. It should also be clear that $\sim$ includes $\diamond$, so $\sim = \diamond$ so $\sim$ must respect $\leq$. □

Corollary 35. *Let Θ be a congruence signature function such that for all a, b if $a \leq b$ then $\Theta(f, k, a) \leq \Theta(f, k, b)$ and $\bowtie$ be some relation family that respects $\leq$. Let $\diamond$ be some relation family that respects $\leq$. The Θ-congruence generated by $\diamond$ is the Θ,$\leq$-congruence generated by $\diamond$*

Once we have a Θ we can construct a our mapping into single relation terms. We define a new term signature that allows us to encode which relation symbol a term is being considered under into the sorts.

Definition 36 (Tagged Signature)**.** *Given some function congruence signature Θ over some term signature $(\mathcal{S}, \Sigma)$ we define a new term signature $(\mathcal{R}, \Sigma^\Theta)$.*

Σ^Θ consists of function symbols of the form f_r where $f \in \Sigma$ and $r \in \mathcal{R}$ where the sort of r and the return sort of f match. These function symbols are sorted according to θ, so if $\text{sort}_\Sigma(f) = ((s_1 \ldots s_n), s)$ then $\text{sort}_{\Sigma^\Theta}(f_r) = ((\Theta(f, 1, r) \ldots \Theta(f, n, r)), r)$.

We then define a way to map between terms in our original signature and terms in our new signature:

Lemma 37. *For each $i \in \mathcal{R}$ we can describe a bijection between $\mathcal{T}(\mathcal{S}, \Sigma)_{\mathrm{sort}_{\mathcal{R}}(i)}$ and $\mathcal{T}(\mathcal{R}, \Sigma^{\Theta})_i$*

$$\mathrm{tag}_{\Theta}(i, f(t_1 \dots t_n)) = f_i(\mathrm{tag}_{\Theta}(\Theta(f, 1, i), t_1) \dots \mathrm{tag}_{\Theta}(\Theta(f, n, i), t_n))$$
$$\mathrm{untag}_{\Theta}(f_i(t_1 \dots t_n)) = f(\mathrm{untag}_{\Theta}(t_1) \dots \mathrm{untag}_{\Theta}(t_n))$$

Example 38.

$$\mathrm{tag}_{\Theta_{\mathbb{Q}}}(\mathrm{eq}, \mathrm{plus}(\mathrm{Qred}(a), b))) = \mathrm{plus}_{\mathrm{eq}}(\mathrm{Qred}_{\mathrm{eq}}(a_{Qeq}), b_{\mathrm{Qeq}})$$
$$\mathrm{untag}_{\Theta_{\mathbb{Q}}}(\mathrm{Qred}_{\mathrm{eq}}(\mathrm{plus}_{\mathrm{Qeq}}(a_{\mathrm{Qeq}}, b_{\mathrm{Qeq}}))) = \mathrm{Qred}(\mathrm{plus}(a, b))$$

Definition 39. *Given a relation family $\sim$ with relation symbols $\mathcal{R}$ over $\mathcal{T}(\mathcal{S}, \Sigma)$, we define a relation $\sim^{\mathrm{tag}_{\Theta}}$ on $\mathcal{T}(\mathcal{R}, \Sigma^{\Theta})$ such that for all $r \in \mathcal{R}, x, y \in \mathcal{T}(\mathcal{R}, \Sigma^{\Theta})_r$*

$$x \sim^{\mathrm{tag}_{\Theta}} y \text{ iff } \mathrm{untag}_{\Theta}(x) \sim_r \mathrm{untag}_{\Theta}(y)$$

Let $\bowtie$ be a well-sorted relation on $\mathcal{T}(\mathcal{R}, \Sigma^{\Theta})$, we define a relation family $\sim^{\mathrm{untag}_{\Theta}}$ with relation symbols $\mathcal{R}$ over $\mathcal{T}(\mathcal{S}, \Sigma)$ such that for all $r \in \mathcal{R}, x, y \in \mathcal{T}(\mathcal{R}, \Sigma)$

$$x \bowtie_r^{\mathrm{untag}_{\Theta}} y \text{ iff } \mathrm{tag}_{\Theta}(r, x) \bowtie \mathrm{tag}_{\Theta}(r, y)$$

It should be clear that $(\sim^{\mathrm{tag}_{\Theta}})^{\mathrm{untag}_{\Theta}} = \sim$ and that $(\bowtie^{\mathrm{untag}_{\Theta}})^{\mathrm{tag}_{\Theta}} = \bowtie$.

Lemma 40. *Given a relation family $\sim$ with relation symbols $\mathcal{R}$ over $\mathcal{T}(\mathcal{S}, \Sigma)$, $\sim$ is a Θ-congruence iff $\sim^{\text{tag}_\Theta}$ is a single relation congruence in $(\mathcal{R}, \Sigma^\Theta)$.*

Equivalently: Given a relation $\bowtie$ on $\mathcal{T}(\mathcal{R}, \Sigma^\Theta)$, $\bowtie$ is a single relation congruence in $(\mathcal{R}, \Sigma^\Theta)$ iff $\bowtie^{\text{untag}_\Theta}$ is a Θ-congruence.

Proof. Given that $\sim$ is a Θ-congruence we need to show that $\sim^{\text{tag}_\Theta}$ is a congruence relation. It should be clear that $\sim^{\text{tag}_\Theta}$ is transitive, reflexive and symmetric, we need to show that for each $f_r \in \Sigma^\Theta$ the following inference is true $\dfrac{s_1 \sim^{\text{tag}_\Theta} t_1 \quad \dots \quad s_n \sim^{\text{tag}_\Theta} t_n}{f_r(s_1 \dots s_n) \sim^{\text{tag}_\Theta} f_r(t_1 \dots t_n)}$ Take some instantiation of the inference where the hypotheses are true. We know by Definition 36 that $\text{sort}_{\Sigma^\Theta}(f_r) = (\Theta(f,1,r) \dots \Theta(f,n,r), r)$, so we can conclude that for each i, $\text{sort}_{\mathcal{T}(\mathcal{R},\Sigma^\Theta)}(s_i) = \Theta(f,i,r)$. Therefore since $s_i \sim^{\text{tag}_\Theta} t_i$ we know that $\text{untag}_\Theta(s_n) \sim_{\Theta(f,n,r)} \text{untag}_\Theta(t_n)$. Because $\sim$ is a Θ-congruence, we can therefore conclude that:

$$f(\text{untag}_\Theta(s_1) \dots \text{untag}_\Theta(s_n)) \sim_r f(\text{untag}_\Theta(t_1) \dots \text{untag}_\Theta(t_n))$$

But because $\text{untag}_\Theta(f_r(s_1 \dots s_n)) = f(\text{untag}_\Theta(s_1) \dots \text{untag}_\Theta(s_n))$ this also tells us that $\text{untag}_\Theta(f_r(s_1 \dots s_n)) \sim_r \text{untag}_\Theta(f_r(t_1 \dots t_n))$ and therefore that $f_r(s_1 \dots s_n) \sim^{\text{tag}_\Theta} f_r(t_1 \dots t_n)$.

In the reverse direction, given that $\sim$ is a congruence relation, we need to show that $\bowtie^{\text{untag}_\Theta}$ is a Θ-congruence. It should be clear that $\bowtie^{\text{untag}_\Theta}$ is an equivalence relation family. Thus we only need to show that the following Horn sentence holds for any $f \in \Sigma$ $r \in \mathcal{R}$.

$$\frac{s_1 \bowtie^{\text{untag}_\Theta}_{\Theta(f,1,r)} t_1 \quad \dots \quad s_n \bowtie^{\text{untag}_\Theta}_{\Theta(f,1,r)} t_n}{f(s_1 \dots s_n) \bowtie^{\text{untag}_\Theta}_r f(t_1 \dots t_n)}$$

Take some instantiation of the sentence where the hypotheses are true. This tells us that for every k, $s_k \bowtie^{\text{untag}_\Theta}_{\Theta(f,k,r)} t_k$ so $\text{tag}_\Theta(\Theta(f,k,r), s_k) \bowtie \text{tag}_\Theta(\Theta(f,k,r), t_k)$. By

congruence this tells us that:

$$f_r(\mathrm{tag}_\Theta(\Theta(f,k,r),s_k)\ldots\mathrm{tag}_\Theta(\Theta(f,k,r),s_k))$$
$$\bowtie f_r(\mathrm{tag}_\Theta(\Theta(f,k,r),t_k)\ldots\mathrm{tag}_\Theta(\Theta(f,k,r),t_k))$$

But this is equivalent to $\mathrm{tag}_\Theta(r, f(s_1\ldots s_n)) \bowtie \mathrm{tag}_\Theta(r, f(t_1\ldots t_n))$ which tells us that $f(s_1\ldots s_n) \bowtie_r^{\mathrm{untag}_\Theta} f(t_1\ldots t_n)$. □

Corollary 41. *Let $\diamond$ be a with relation symbols $\mathcal{R}$ over $\mathcal{T}(\mathcal{S},\Sigma)$.*

If $\sim$ is the congruence in $(\mathcal{R},\Sigma^\Theta)$ generated by $\diamond^{\mathrm{tag}_\Theta}$, than $\sim^{\mathrm{untag}_\Theta}$ is the Θ-congruence generated by $\diamond$.

Theorem 42. *Given some congruence signature $\mathcal{C}$ such that $\mathcal{M}(\mathcal{C})$ is resolvable, let Θ be the induced congruence signature function as described in Definition 31.*

Let $\diamond$ be some relation family that respects $\leq$ some relation family representing a set of ground equations.

If $\sim$ is the congruence generated by $\diamond^{\mathrm{tag}_\Theta}$, than the $\mathcal{C},\leq$-congruence generated by $\diamond$ is $\sim^{\mathrm{untag}_\Theta}$.

Proof. By Corollary 41, $\sim^{\mathrm{untag}_\Theta}$ is the Θ-congruence generated by $\diamond$, which by Corollary 35 $\sim^{\mathrm{untag}_\Theta}$ is the $\Theta,\leq$-congruence generated by $\diamond$, which by Corollary 33 is the $\mathcal{M}(\mathcal{C}),\leq$-congruence generated by $\diamond$ which by Corollary 28 is the $\mathcal{C},\leq$-congruence generated by $\diamond$. □

Corollary 43. *Under the same conditions as Theorem 42,*

If $\diamond \models_{\mathcal{C},\leq} x \approx_r y$ if and only if $\diamond^{\mathrm{tag}_\Theta} \models \mathrm{tag}_\Theta(r,x) \approx \mathrm{tag}_\Theta(r,y)$

Proof. let $\sim$ be the congruence relation generated by $\diamond^{\mathrm{tag}_\Theta}$. By Theorem 42 the $\mathcal{C},\leq$-congruence generated by $\diamond$ is $\sim^{\mathrm{untag}_\Theta}$, so $\diamond \models_{\mathcal{C},\leq} x \approx_r y$ if and only if $x \sim_r^{\mathrm{untag}_\Theta} y$ if

and only if $\mathrm{tag}_\Theta(r,x) \sim \mathrm{tag}_\Theta(r,y)$ if and only if $\diamond^{\mathrm{tag}_\Theta} \models \mathrm{tag}_\Theta(r,x) \approx \mathrm{tag}_\Theta(r,y)$. $\square$

Chapter 4

Implementation

The congruence tactic is implemented as a Coq plugin. This means that it is written using separate OCaml code that coq is instructed to load. Our tactic is on the other hand was implemented in Ltac2, which can be directly embedded in coq source files. Ltac2 is the the successor to Ltac1, the original tactic language for coq. While Ltac1 is a very powerful tool for proof authoring, it lacks features common to most general purpose langues. This makes it very hard to implement algorithms in Ltac1. Ltac2 on the other hand is built on top of ML and allows many algorithms to be directly implemented, to the point where building a full congruence closure implementation becomes feasible. We opted to implement our tactic in Ltac2 rather than as a plugin for two major reasons:

- Ltac2 is relatively new, having been added with Coq version 8.11.0 in January of 2020. As such this is an interesting look into what the next generation of Coq tactics will look like.
- As most of the primitives from Ltac1 are available in Ltac2, if one is already familiar with Ltac1, Ltac2 will be much easier to learn than the Coq plugin ecosystem. Particularly, Ltac1 can be called directly from Ltac2, which is

a shortcut we use in a number of cases to delegate tasks to Coq's existing systems for setoids.

With this in mind, beyond basic functionality there are two major goals we for our tactic implementation:

- Create a system with re-usable components, as the Coq Ltac2 tool-set is still developing, this may allow some components to be reused even if some components need to be re-designed to take advantage of new developments. Specifically in our case, while the congruence closure implementation is built to be separate from the code concerned with coq proof states.
- As accurately as possible mirror algorithms as they would be done in a typical programming language.

4.1 Components

4.1.1 UFProof.v

This file provides type definitions to abstractly represent an equality proof with a polymorphic parameter for the type of "axioms". The type it provides doesn't provide for congruence axioms, rather it only provides constructors for transitivity, symmetry, reflexivity and axioms. Congruence is later accomplished by being added as a axiom.

Though this type does implement minor proof cleaning logic, specifically that symmetry is involutive, and reflexivity is the identity element on transitivity, it does not attempt to reduce proofs to any sort of a canonical form, and the tactic makes no guarantees about the exact from of the proof. For the most part this type exists

as an intermediate structure to defer Coq typechecking of proof terms until the end and limit it to actually used proofs.

4.1.2 UnionFind.v

This implements a union find data structure, following the algorithm described by Nelson and Oppen [NO80] with a few important modifications.

1. Each vertex that is not a representative holds a CCProof that is is equal to its parent.

2. There is a polymorphic parameter for information stored with the representative of each class. For CongruenceClosure this is the left and right ancestors, but this is again left flexible in the interest of keeping components re-usable.

4.1.3 CongruenceClosure.v

This file implements binary congruence closure over an abstract graph. It follows the Nelson and Oppen [NO80] algorithm as described in Term Rewriting and All That [BN99] with the modification that it replaces the recursive call to merge with appending to a separate stack. The intent of this was simply to make debugging easier.

The input to this file is a term graph, represented as an array of optional pairs of left and right sub-term indexes, and a list of equations with CCProofs. The output is a union find data structure that is the smallest congruence over the term graph containing the given equations.

4.1.4 CCBuilder.v

CCBuilder is a wrapper layer around congruence closure that helps with converting terms into term graphs, and tracking equalities on those terms.

It provides a data type CCBuilder that supports the following operations:

1. Insert a new term with the given left and right subterms, returning its id
2. Insert a new term treated as a constant symbol
3. Query the existing terms and return the allocated id. If the query doesn't match, call a callback function that can allocate an id.
4. Record a ground equation between two terms that have been given ids
5. Record a 'Goal' pair / disequality, ie a pair of terms that if proven congruent, would mean that the tactic has succeeded.
6. Run the cc builder, returning the goal pair that is implied by all registered ground equations.

Unfortunately, this is where a steep price is payed for the environment this tactic is built in. At the time of writing, Ltac2 lacks any way to hash terms, nor integer modulo, both of which would be needed to implement this with a fast data structure. As such our query operations are limited to $O(n)$.

4.1.5 OverEq.v

This file is not used in the final tactic, but serves as a testing ground for the congruence closure logic. As the name implies, it implements congruence over Logic.eq, using the helpers provided by CCBuilder.v. It consists of three parts:

1. Logic to convert proofs of type CCProof to actual Coq proofs. Mapping the different constructors onto different Coq lemmas.

2. A function 'addTerm' that takes a Coq term and adds it (if necessary) to a CCBuilder, returning the allocated id.

3. A function 'addEquations' that records all ground equations in the proof state in a cc builder

4. A function 'addGoal' which adds the goal as a disequality to the CCBuilder.

From there some minimal boilerplate combines these together to produce a fully Ltac2 congruence tactic.

4.1.6 OverSetoid.v

OverSetoid houses the actual tactic implementations. It contains all of the same parts as OverEq.v, except that they now follow the construction described in our theory. The most important part of this being the computation of Θ. The computation of Θ is largely delegated to Coq's existing setoid systems. Specifically the value chosen for $\Theta(f, k, r)$ is the first r' in the array such that setoid_rewrite followed by reflexivity can construct a proof of the simplified congruence signature entry's induced axiom. It is worth noting that we do not generalize over the first $k-1$ arguments to f, rather we re-use the values already present in the proof state. This allows us to avoid the choice of what is considered function symbol and what is considered arguments. Specifically if we can't compute a valid value for Θ we treat the term we where trying to subdivide as a function symbol.

4.2 A short manual

The setoid_congruence tactic works as an automatic setoid_rewrite finisher tactic. As its arguments it takes a list of relations in *decreasing* order of size (this condition is not checked) to consider. The tactic will then find a chain of setoid_rewrites followed by a reflexivity that can solve the current goal.

Limitations:

1. Unlike setoid_rewrite This tactic can not rewrite terms under binders.

2. This tactic is only guaranteed to find a solution if the relations provided over each type are well ordered with respect to subrelation

Note you do not need to list every relation you are using, the tactic will use subrelation instance to find the smallest relations in your list that contains each hypothesis, and the largest relation in your list sufficient to prove the goal.

Chapter 5

Conclusions

A natural area of future work would be to relax the condition on our theories that we assume. While I didn't explore the topic, it seems that it would be possible to do this by introducing additional relation symbols. Specifically I if you could extend the set of relation symbols to be the power-set of the original relations, where each relation is interpreted as be the equivalence relation generated by the union of it's members. We believe that this would always result in a resolvable congruence signature, but an algorithm that takes this approach would quickly become the victim of combinatorial explosion.

The implementation also has a lot of possible improvements. A more mature implementation would check its conditions. Additionally future extensions to the Ltac2 standard library could result in massive speed improvements. For example, at the time of writing Ltac2 doesn't include integer division or modulo, which would be needed to implement the faster algorithm described by Downey, Sethi and Tarjan [DST80]

Bibliography

[Ack54] W. Ackermann. *Solvable Cases of the Decision Problem*. Janua linguarum: Series maior. North-Holland, 1954.

[BN99] Franz Baader and Tobias Nipkow. *Term rewriting and all that*. Cambridge university press, 1999.

[CLS96] David Cyrluk, Patrick Lincoln, and Natarajan Shankar. On Shostak's decision procedure for combinations of theories. In M. A. McRobbie and J. K. Slaney, editors, *Automated Deduction — Cade-13*, pages 463–477, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[coq] *The Coq Proof Assistant*. `https://coq.inria.fr/`.

[Cor01] Pierre Corbineau. Autour de la clôture de congruence avec Coq. Master's thesis, Université Paris, 2001.

[Cor06] Pierre Corbineau. Deciding equality in the constructor theory. In *International Workshop on Types for Proofs and Programs*, pages 78–92. Springer, 2006.

[DST80] Peter J Downey, Ravi Sethi, and Robert Endre Tarjan. Variations on the common subexpression problem. *Journal of the ACM (JACM)*, 27(4):758–771, 1980.

[Koz77] Dexter Kozen. Complexity of finitely presented algebras. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, STOC '77, page 164–177, New York, NY, USA, 1977. Association for Computing Machinery.

[NO80] Greg Nelson and Derek C Oppen. Fast decision procedures based on congruence closure. *Journal of the ACM (JACM)*, 27(2):356–364, 1980.

[NO05] Robert Nieuwenhuis and Albert Oliveras. Proof-producing congruence closure. In Jürgen Giesl, editor, *Term Rewriting and Applications*, pages 453–468, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[Sho78] Robert E. Shostak. An algorithm for reasoning about equality. *Commun. ACM*, 21(7):583–585, July 1978.

[Sho82] Robert E Shostak. Deciding combinations of theories. In *International Conference on Automated Deduction*, pages 209–222. Springer, 1982.

[Soz] Matthieu Sozeau. *Generalized rewriting - Coq 8.13.2 documentation*. `https://coq.inria.fr/refman/addendum/generalized-rewriting.html`.