Congruence Closure in Intensional Type Theory

Daniel Selsam[1] Leonardo de Moura[2]

[1]Stanford University

[2]Microsoft Research

June 30, 2016

Goal

- Intensional type theory (ITT)
  - Coq, Lean, Agda, Epigram, Idris
- Many striking successes.
  - CompCert, FSCQ, Odd-order theorem
- Major ongoing work.
  - MIT, Princeton, UPenn, Yale, Cornell, CMU, Tulane, others
- All this despite very little automation.
  - Workarounds: manual proving, custom tactic scripts, restricting to limited subsets of ITT.
- **Goal: an automated theorem prover for ITT.**

Congruence Closure

- The most glaring limitation: no algorithm for congruence closure.

- Congruence closure (CC) is fundamental to SMT.
  - link between SAT and theory solvers
  - part of other theory solvers
  - guides heuristic instantiation
  - models

- We want CC to be a core component of our prover for ITT.

Outline

Review
- Congruence closure
- Intensional type theory (ITT)

Equality in ITT: the source of the problem
- Homogeneous equality
- Heterogeneous equality

Congruence in ITT
- Heterogeneous equality is not a congruence relation
- New notion of congruence for heterogeneous equality

Congruence Closure in ITT
- Congruence closure template
- New congruence closure procedure for ITT

First steps to a prover

What is congruence closure?

- Congruence closure is an efficient decision procedure for equality.
- Examples:
  - Input: $\texttt{a} = \texttt{b} \wedge \texttt{f a} \neq \texttt{f b}$
    Output: UNSAT
  - Input: $\texttt{a} = \texttt{b} \wedge \texttt{f a} = \texttt{g b} \wedge \texttt{f b} \neq \texttt{g b}$
    Output: UNSAT
- Functions and constants "uninterpreted".
- Curry notation: `f a` means `f(a)`

Review of standard congruence closure

- Inputs: $\mathtt{a} = \mathtt{b}$, $\mathtt{f}\ \mathtt{a} = \mathtt{g}\ \mathtt{b}$, $\mathtt{f}\ \mathtt{b} \neq \mathtt{g}\ \mathtt{b}$

Review of standard congruence closure

- Inputs: `a` = `b`, `f a` = `g b`, `f b` ≠ `g b`

0 initial equivalence classes: {`a`} {`b`} {`f a`} {`f b`} {`g b`}

Review of standard congruence closure

- Inputs: `a = b`, `f a = g b`, `f b` $\neq$ `g b`

0 initial equivalence classes: {`a`} {`b`} {`f a`} {`f b`} {`g b`}

1 `merge(a, b)`: {`a`, `b`} {`f a`} {`f b`} {`g b`}

Review of standard congruence closure

- Inputs: `a = b`, `f a = g b`, `f b` $\neq$ `g b`

0 initial equivalence classes: {`a`} {`b`} {`f a`} {`f b`} {`g b`}

1 `merge(a, b)`: {`a`, `b`} {`f a`} {`f b`} {`g b`}

2 check terms that use `a` for new congruences
  - `f a` is now congruent to `f b`
  - `merge(f a, f b)`: {`a`, `b`} {`f a`, `f b`} {`g b`}

Review of standard congruence closure

- Inputs: `a = b`, `f a = g b`, `f b` $\neq$ `g b`

0 initial equivalence classes: {`a`} {`b`} {`f a`} {`f b`} {`g b`}

1 `merge(a, b)`: {`a`, `b`} {`f a`} {`f b`} {`g b`}

2 check terms that use `a` for new congruences
  - `f a` is now congruent to `f b`
  - `merge(f a, f b)`: {`a`, `b`} {`f a`, `f b`} {`g b`}

3 `merge(f a, g b)`: {`a`, `b`} {`f a`, `f b`, `g b`}

Review of standard congruence closure

- Inputs: `a = b`, `f a = g b`, `f b` $\neq$ `g b`

0 initial equivalence classes: {`a`} {`b`} {`f a`} {`f b`} {`g b`}

1 `merge(a, b)`: {`a`, `b`} {`f a`} {`f b`} {`g b`}

2 check terms that use `a` for new congruences
  - `f a` is now congruent to `f b`
  - `merge(f a, f b)`: {`a`, `b`} {`f a`, `f b`} {`g b`}

3 `merge(f a, g b)`: {`a`, `b`} {`f a`, `f b`, `g b`}

- Conclude UNSAT, since `f b = g b` and `f b` $\neq$ `g b`

What is ITT?

- The main feature of ITT: *dependent types*.
- New construct: `Π (x : A), B x`
  - Like `A → B`, but can refer to argument `x` in return type
- Simply-typed lists
  - `list.all_zeros : ℕ → list ℕ`
  - `list.all_zeros 0 : list ℕ`
  - `list.tail (list.all_zeros 0) : list ℕ`
  - runtime error
- Dependently-typed lists
  - `ilist.all_zeros : Π n : ℕ, ilist ℕ n`
  - `ilist.all_zeros 0 : ilist ℕ 0`
  - `ilist.tail (ilist.all_zeros 0)` gives a type error

Dependent types

- Deceptively powerful idea: get theorems "for free".
  - `ilist A n`
    - prevents *list index out of range*
  - `rbtree color n`
    - type guarantees red-black tree is well-formed
  - `f : ∏ a:A, Pre_f a → B a`
    - prevents *invalid input*
  - `prog : HoareState Pre Post`
    - bundle stateful computation as a Hoare triple
- Methodology: correct *by construction*
- Cost: some complications
  - most notably: equality is no longer a congruence relation!

Our contribution

- A weaker notion of congruence that holds in ITT.
- An efficient and proof-producing congruence closure procedure built on it.
- First step towards a prover.

Outline

Review
- Congruence closure
- Intensional type theory (ITT)

Equality in ITT: the source of the problem
- Homogeneous equality
- Heterogeneous equality

Congruence in ITT
- Heterogeneous equality is not a congruence relation
- New notion of congruence for heterogeneous equality

Congruence Closure in ITT
- Congruence closure template
- New congruence closure procedure for ITT

First steps to a prover

Equality is subtle in ITT

- Congruence is subtle because equality is subtle.

- Different kinds of equality.
  1. homogeneous equality:
     - congruence relation but no dependent types
  2. heterogeneous equality:
     - dependent types but no longer a congruence relation

- Warning: may be confusing at first.

1. Homogeneous equality

- eq $\mathtt{A}$ $\mathtt{a}_1$ $\mathtt{a}_2$ means $\mathtt{a}_1$: $\mathtt{A}$ and $\mathtt{a}_2$: $\mathtt{A}$ are (homogeneously) equal.
  - notation: $\mathtt{a}_1$ = $\mathtt{a}_2$
- *homogeneous* since $\mathtt{a}_1$ and $\mathtt{a}_2$ must have the same type.
- eq behaves "as you would expect".
  - $\mathtt{P}$ $\mathtt{a}_1$ and $\mathtt{a}_1$ = $\mathtt{a}_2$ implies $\mathtt{P}$ $\mathtt{a}_2$.

Limitations of homogeneous equality

- Given: `f : Π a:A, B a`
- Cannot even *state* the congruence property for `f`:

  `eq A` a_1 $a_2 \rightarrow$ `eq ?` `(f` a_1`) (f` a_2`)`
- Problem:

  `f` a_1 `: B` a_1
  `f` a_2 `: B` a_2

3. Heterogeneous equality

- `heq A B a b` means `a : A` and `b : B` are (heterogenously) equal.
  - notation: `a == b`
- Can always state `a == b`.
- Meaning is subtle:
  - $\exists$ `pf : (A = B), cast a pf = b`
  - `heq` hides the casts
  - when types are the same: $\mathtt{a}_1 \mathtt{ == } \mathtt{a}_2 \rightarrow \mathtt{a}_1 \mathtt{ = } \mathtt{a}_2$
- `heq` is weaker than `eq`
  - no longer a congruence relation

Outline

Review
- Congruence closure
- Intensional type theory (ITT)

Equality in ITT: the source of the problem
- Homogeneous equality
- Heterogeneous equality

Congruence in ITT
- Heterogeneous equality is not a congruence relation
- New notion of congruence for heterogeneous equality

Congruence Closure in ITT
- Congruence closure template
- New congruence closure procedure for ITT

First steps to a prover

Congruence in ITT

- Homogeneous equality is a congruence relation:

```
f = f′
a = a′
----------------
⊢ f a = f′ a′
```

- Heterogeneous equality is **not** a congruence relation:

```
f == f′
a == a′
----------------
⊬ f a == f′ a′
```

- Our approach: find a weaker notion of congruence for `heq`.

n-ary congruence by repeated binary congruence

Suppose we want to prove:

$$\begin{array}{l} \mathtt{f} = \mathtt{f}' \\ \mathtt{a}_1 = \mathtt{a}'_1 \\ \mathtt{a}_2 = \mathtt{a}'_2 \\ \hline \vdash \mathtt{f}\ \mathtt{a}_1\ \mathtt{a}_2 = \mathtt{f}'\ \mathtt{a}'_1\ \mathtt{a}'_2 \end{array}$$

n-ary congruence by repeated binary congruence

Suppose we want to prove:

$$\begin{array}{l} \mathtt{f} = \mathtt{f}' \\ \mathtt{a}_1 = \mathtt{a}'_1 \\ \mathtt{a}_2 = \mathtt{a}'_2 \\ \text{----------} \\ \vdash \mathtt{f}\ \mathtt{a}_1\ \mathtt{a}_2 = \mathtt{f}'\ \mathtt{a}'_1\ \mathtt{a}'_2 \end{array}$$

- Step 1: $\mathtt{f}\ \mathtt{a}_1 = \mathtt{f}'\ \mathtt{a}'_1$

n-ary congruence by repeated binary congruence

Suppose we want to prove:

$$\begin{array}{l} \mathtt{f} = \mathtt{f}' \\ \mathtt{a}_1 = \mathtt{a}'_1 \\ \mathtt{a}_2 = \mathtt{a}'_2 \\ \hline \vdash \mathtt{f}\ \mathtt{a}_1\ \mathtt{a}_2 = \mathtt{f}'\ \mathtt{a}'_1\ \mathtt{a}'_2 \end{array}$$

- Step 1: $\mathtt{f}\ \mathtt{a}_1 = \mathtt{f}'\ \mathtt{a}'_1$
- Step 2: $(\mathtt{f}\ \mathtt{a}_1)\ \mathtt{a}_2 = (\mathtt{f}'\ \mathtt{a}'_1)\ \mathtt{a}'_2$

n-ary congruence lemmas

- No binary congruence rule for heterogeneous equality.

$$\frac{\mathtt{f} \mathrel{==} \mathtt{f}' \quad \mathtt{a}_1 \mathrel{==} \mathtt{a}'_1 \quad \mathtt{a}_2 \mathrel{==} \mathtt{a}'_2}{\not\forall\ \mathtt{f}\ \mathtt{a}_1\ \mathtt{a}_2 \mathrel{==} \mathtt{f}'\ \mathtt{a}'_1\ \mathtt{a}'_2}$$

n-ary congruence lemmas

- No binary congruence rule for heterogeneous equality.

$$\frac{\mathtt{f} == \mathtt{f}' \quad \mathtt{a}_1 == \mathtt{a}'_1 \quad \mathtt{a}_2 == \mathtt{a}'_2}{\not\forall\ \mathtt{f}\ \mathtt{a}_1\ \mathtt{a}_2 == \mathtt{f}'\ \mathtt{a}'_1\ \mathtt{a}'_2}$$

- Consolation: new atomic rule for arity 2 ($\mathtt{hcongr}_2$)

$$\frac{\mathtt{f} = \mathtt{f}' \quad \mathtt{a}_1 == \mathtt{a}'_1 \quad \mathtt{a}_2 == \mathtt{a}'_2}{\vdash\ \mathtt{f}\ \mathtt{a}_1\ \mathtt{a}_2 == \mathtt{f}'\ \mathtt{a}'_1\ \mathtt{a}'_2}$$

New notion of congruence for heterogeneous equality

- Idea: prove custom congruence lemmas for every arity
- Example ($\texttt{hcongr}_3$):

$$\begin{array}{l} \texttt{f} = \texttt{f}' \\ \texttt{a}_1 == \texttt{a}'_1 \\ \texttt{a}_2 == \texttt{a}'_2 \\ \texttt{a}_3 == \texttt{a}'_3 \\ \texttt{----------} \\ \vdash \texttt{f a}_1\ \texttt{a}_2\ \texttt{a}_3 == \texttt{f}'\ \texttt{a}'_1\ \texttt{a}'_2\ \texttt{a}'_3 \end{array}$$

- General rule: $\texttt{hcongr}_k$ proves that two terms are `heq` provided:
  1. the final k arguments are each `heq`, and
  2. the remaining prefixes are `eq`,

New notion of congruence for heterogeneous equality

- Reminder: $\texttt{hcongr}_k$ proves that two terms are heq provided:
  1. the final k arguments are each heq, and
  2. the remaining prefixes are eq,

- New definition: two terms are *congruent* if they can be proved heq by $\texttt{hcongr}_k$ for some k.

- Example: $\texttt{f}\ \texttt{a}_1\ \texttt{a}_2\ \texttt{a}_3\ \texttt{=?=}\ \texttt{g}\ \texttt{b}_1\ \texttt{b}_2\ \texttt{b}_3\ \texttt{b}_4$ congruent by
  - $\texttt{hcongr}_1$ if $\texttt{f}\ \texttt{a}_1\ \texttt{a}_2\ \texttt{a}_3\ \texttt{=?=}\ \texttt{g}\ \texttt{b}_1\ \texttt{b}_2\ \texttt{b}_3\ \texttt{b}_4$
  - $\texttt{hcongr}_2$ if $\texttt{f}\ \texttt{a}_1\ \texttt{a}_2\ \texttt{a}_3\ \texttt{=?=}\ \texttt{g}\ \texttt{b}_1\ \texttt{b}_2\ \texttt{b}_3\ \texttt{b}_4$
  - $\texttt{hcongr}_3$ if $\texttt{f}\ \texttt{a}_1\ \texttt{a}_2\ \texttt{a}_3\ \texttt{=?=}\ \texttt{g}\ \texttt{b}_1\ \texttt{b}_2\ \texttt{b}_3\ \texttt{b}_4$

  $\texttt{prefixes= args==}$

Outline

Review
- Congruence closure
- Intensional type theory (ITT)

Equality in ITT: the source of the problem
- Homogeneous equality
- Heterogeneous equality

Congruence in ITT
- Heterogeneous equality is not a congruence relation
- New notion of congruence for heterogeneous equality

Congruence Closure in ITT
- Congruence closure template
- New congruence closure procedure for ITT

First steps to a prover

Congruence closure template

- Want: a CC procedure for our new notion of congruence.

- Two main requirements:
  1. tell if two terms are congruent
  2. `parents(t)`: enumerate terms that might trigger a new congruence for `t`

Traditional congruence closure

1. tell if two terms are congruent
   - `f a` and `g b` are congruent iff `f = g` and `a = b`.

2. `parents(t)`: enumerate terms that might trigger a new congruence for `t`
   - parents of `f a` are `f` and `a`

Congruence closure over heterogeneous equality

[1] tell if two terms are congruent

- Reminder: two terms are congruent iff there is some k such that:
  1. the final k arguments are each `heq`, and
  2. the remaining prefixes are `eq`,

- Running example: $\mathtt{f}\ \mathtt{a}_1\ \mathtt{a}_2\ \mathtt{a}_3\ \mathtt{=?=}\ \mathtt{g}\ \mathtt{b}_1\ \mathtt{b}_2\ \mathtt{b}_3\ \mathtt{b}_4$ congruent by
  - $\mathtt{hcongr}_1$ if $\mathtt{f}\ \mathtt{a}_1\ \mathtt{a}_2\ \mathtt{a}_3\ \mathtt{=?=}\ \mathtt{g}\ \mathtt{b}_1\ \mathtt{b}_2\ \mathtt{b}_3\ \mathtt{b}_4$
  - $\mathtt{hcongr}_2$ if $\mathtt{f}\ \mathtt{a}_1\ \mathtt{a}_2\ \mathtt{a}_3\ \mathtt{=?=}\ \mathtt{g}\ \mathtt{b}_1\ \mathtt{b}_2\ \mathtt{b}_3\ \mathtt{b}_4$
  - $\mathtt{hcongr}_3$ if $\mathtt{f}\ \mathtt{a}_1\ \mathtt{a}_2\ \mathtt{a}_3\ \mathtt{=?=}\ \mathtt{g}\ \mathtt{b}_1\ \mathtt{b}_2\ \mathtt{b}_3\ \mathtt{b}_4$

  `prefixes= args==`

- Recursive algorithm: `f a` and `g b` are congruent if and only if `a == b` and either
  - `f = g` ($\mathtt{hcongr}_1$)
  - `f` and `g` are congruent recursively ($\mathtt{hcongr}_{k+1}$)

Congruence closure over heterogeneous equality

[2] `parents(t)`: enumerate terms that might trigger a new congruence for `t`

- Running example: $\mathtt{f}\ \mathtt{a}_1\ \mathtt{a}_2\ \mathtt{a}_3\ \mathtt{=?=}\ \mathtt{g}\ \mathtt{b}_1\ \mathtt{b}_2\ \mathtt{b}_3\ \mathtt{b}_4$ congruent by
  - $\mathtt{hcongr}_1$ if $\mathtt{f}\ \mathtt{a}_1\ \mathtt{a}_2\ \mathtt{a}_3\ \mathtt{=?=}\ \mathtt{g}\ \mathtt{b}_1\ \mathtt{b}_2\ \mathtt{b}_3\ \mathtt{b}_4$
  - $\mathtt{hcongr}_2$ if $\mathtt{f}\ \mathtt{a}_1\ \mathtt{a}_2\ \mathtt{a}_3\ \mathtt{=?=}\ \mathtt{g}\ \mathtt{b}_1\ \mathtt{b}_2\ \mathtt{b}_3\ \mathtt{b}_4$
  - $\mathtt{hcongr}_3$ if $\mathtt{f}\ \mathtt{a}_1\ \mathtt{a}_2\ \mathtt{a}_3\ \mathtt{=?=}\ \mathtt{g}\ \mathtt{b}_1\ \mathtt{b}_2\ \mathtt{b}_3\ \mathtt{b}_4$

  `prefixes= args==`

- parents of $\mathtt{f}\ \mathtt{a}_1\ \ldots\ \mathtt{a}_n$: all of its strict subterms (i.e. prefixes and arguments).

Outline

Review
- Congruence closure
- Intensional type theory (ITT)

Equality in ITT: the source of the problem
- Homogeneous equality
- Heterogeneous equality

Congruence in ITT
- Heterogeneous equality is not a congruence relation
- New notion of congruence for heterogeneous equality

Congruence Closure in ITT
- Congruence closure template
- New congruence closure procedure for ITT

First steps to a prover

Heuristic instantiation

- Congruence closure on its own is of limited value.
  - Only operates on ground facts, no quantifiers
- CC cannot prove: $(\forall$ `x, f x = x)` $\vdash$ `g (f a) = g a`
  - solution: instantiate at `a`
  - CC can prove: `f a = a` $\vdash$ `g (f a) = g a`
- Congruence closure + heuristic instantiation is already very powerful.
- E-matching
  - simple idea but details out of scope

Congruence closure for length-indexed lists

- Length-indexed lists (`ilist A n`) often avoided in big projects.
  - lack of automation has outweighed benefits
- Our CC procedure works just as well on `ilists`.
- Induction + E-matching + congruence closure can prove:

```
lemma rev_app : ∀ (A : Type) (n₁ n₂ : ℕ)
  (ℓ₁ : ilist A n₁) (ℓ₂ : ilist A n₂),
    rev (ℓ₁ ++ ℓ₂) == (rev ℓ₂) ++ (rev ℓ₁)
```

Congruence closure for safe arithmetic

- Another use of dependent types: safe arithmetic

```
safe_log : Π (x : ℝ), x > 0 → ℝ
safe_inv : Π (x : ℝ), x ≠ 0 → ℝ
```

- Define
  - `log x` := `safe_log x p`
  - $\texttt{y}^{-1}$:= `safe_inv y q`
- E-matching + CC can solve:

```
∀ (x y z w : ℝ), x > 0 → y > 0 → z > 0
  → w > 0 → x * y = exp z + w
  → log (2 * w * exp z + w^2 + exp (2 * z)) / -2
      = log y^-1 - log x
```

Summary

- New notion of congruence for heterogeneous equality in ITT.
- New efficient and proof-producing congruence closure procedure for ITT that supports dependent types.
- CC is the core of our new prover for ITT.
  - long-term effort led by Leonardo de Moura at MSR
  - built into Lean
  - CC + E-matching: very useful already