Programming up to Congruence

Vilhelm Sjöberg and Stephanie Weirich

University of Pennsylvania

October 14, 2013

WG 2.8 Aussois, France

FP + dependent types

What does this mean?

Goal

A functional programming language with an expressive type system, with extended capabilities for “lightweight” verification

Requirements:

- Core language for functional programming (including nontermination)
- Full-spectrum dependency
- Erasable arguments (both types and values)
- Extrinsic semantics (type annotations don’t matter)

Nongoal: mathematical foundations, full program verification

Plan of attack

1. Design explicitly-typed **core language** that defines the semantics. (Like FC, core language has explicit type coercions.)
2. Design a declarative specification of a **surface language**, which specifies what type annotations and coercions can be omitted.
   - Bidirectional type checking
   - **Congruence closure**
3. Figure out how to implement the declarative system through elaboration into the core language.

Core language

Core language

$$\begin{array}{llcl} \text{expressions} & a, b, c, A, B & ::= & \mathsf{Type} \mid x \mid \mathsf{rec}\, f_A.v \\ & & \mid & (x:A) \to B \mid \lambda x_A.a \mid a\ b \\ & & \mid & a = b \mid \mathsf{join}_\sigma \mid a_{\triangleright v} \mid \ldots \\ \text{coercion} & \sigma & ::= & \ldots \end{array}$$

Type annotations are optional, ignored by operational semantics, and removed by $|a|$ notation.

- "Call-by-value Cayenne"
- Fragment of [Sjöberg et al., MSFP'12], which is in turn a fragment of ZOMBIE core [Casinghino et al. POPL'14]

Type system

$\boxed{\Gamma \vdash a : A}$

$$\frac{\vdash \Gamma}{\Gamma \vdash \mathsf{Type} : \mathsf{Type}}$$

$$\frac{x : A \in \Gamma \quad \vdash \Gamma}{\Gamma \vdash x : A}$$

$$\frac{\Gamma \vdash A : \mathsf{Type} \quad \Gamma, x : A \vdash B : \mathsf{Type}}{\Gamma \vdash (x{:}A) \to B : \mathsf{Type}}$$

$$\frac{\Gamma, f : A \vdash v : A \quad \Gamma \vdash A : \mathsf{Type} \quad A \text{ is } (x{:}A_1) \to A_2}{\Gamma \vdash \mathsf{rec}\ f_{A}.v : A}$$

$$\frac{\Gamma, x : A \vdash b : B}{\Gamma \vdash \lambda x_{A}.b : (x{:}A) \to B}$$

$$\frac{\Gamma \vdash a : A \to B \quad \Gamma \vdash b : A}{\Gamma \vdash a\ b : B}$$

$$\frac{\Gamma \vdash a : (x{:}A) \to B \quad \Gamma \vdash v : A}{\Gamma \vdash a\ v : \{v/x\}B}$$

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B}{\Gamma \vdash a = b : \mathsf{Type}}$$

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash v : A = B \quad \Gamma \vdash B : \mathsf{Type}}{\Gamma \vdash a_{\triangleright v} : B}$$

When are expressions equal?

- When they evaluate the same way

$$\frac{|a| \leadsto^{i}_{\mathsf{cbv}} a' \quad |b| \leadsto^{j}_{\mathsf{cbv}} a' \quad \Gamma \vdash a = b : \mathsf{Type}}{\Gamma \vdash \mathsf{join}_{\leadsto_{\mathsf{cbv}} i\, j : a = b} : a = b}$$

When are expressions equal?

- When they evaluate the same way

$$\frac{|a| \rightsquigarrow^{i}_{\mathsf{cbv}} a' \quad |b| \rightsquigarrow^{j}_{\mathsf{cbv}} a' \quad \Gamma \vdash a = b : \mathsf{Type}}{\Gamma \vdash \mathsf{join}_{\rightsquigarrow_{\mathsf{cbv}} i\, j : a = b} : a = b}$$

- When their subcomponents are equal (congruence)

$$\frac{\overline{\Gamma \vdash v_j : a_j = b_j}^{\,j} \quad \Gamma \vdash \overline{\{a_j/x_j\}}^{\,j}\, c = \overline{\{b_j/x_j\}}^{\,j}\, c : \mathsf{Type}}{\Gamma \vdash \mathsf{join}_{\overline{\{\sim v_j/x_j\}}^{\,j}\, c} : \overline{\{a_j/x_j\}}^{\,j}\, c = \overline{\{b_j/x_j\}}^{\,j}\, c}$$

When are expressions equal?

- When they evaluate the same way

$$\frac{|a| \leadsto^{i}_{\mathsf{cbv}} a' \quad |b| \leadsto^{j}_{\mathsf{cbv}} a' \quad \Gamma \vdash a = b : \mathsf{Type}}{\Gamma \vdash \mathsf{join}_{\leadsto_{\mathsf{cbv}}\, i\, j : a = b} : a = b}$$

- When their subcomponents are equal (congruence)

$$\frac{\overline{\Gamma \vdash v_j : a_j = b_j}^{\,j} \quad \Gamma \vdash \overline{\{a_j/x_j\}}^{\,j}\, c = \overline{\{b_j/x_j\}}^{\,j}\, c : \mathsf{Type}}{\Gamma \vdash \mathsf{join}_{\overline{\{\sim v_j/x_j\}}^{\,j}\, c} : \overline{\{a_j/x_j\}}^{\,j}\, c = \overline{\{b_j/x_j\}}^{\,j}\, c}$$

- Reflexivity, symmetry and transitivity are derivable

$$\frac{\Gamma \vdash v : a = b}{\Gamma \vdash \mathsf{join}_{\leadsto_{\mathsf{cbv}}\, b = b \,\triangleright\, \mathsf{join}_{\sim v = b}} : b = a}$$

Surface language

Inferring λ annotations: Bidirectional type system

Can we infer type annotations, such as $\mathsf{rec}\, f_A.a$ and $\lambda x_A.a$?

$$\boxed{\Gamma \vdash a \Rightarrow A} \qquad \boxed{\Gamma \vdash a \Leftarrow A}$$

$$\frac{x : A \in \Gamma}{\Gamma \vdash x \Rightarrow A}$$

$$\frac{\Gamma, x : A \vdash b \Leftarrow B}{\Gamma \vdash \lambda x.a \Leftarrow (x{:}A) \rightarrow B}$$

$$\frac{\Gamma \vdash a \Rightarrow (x{:}A) \rightarrow B \qquad \Gamma \vdash v \Leftarrow A}{\Gamma \vdash a\ v \Rightarrow \{v/x\}B}$$

$$\frac{\Gamma \vdash A \Leftarrow \mathsf{Type} \qquad \Gamma, f : A \vdash v \Leftarrow A \qquad A = (x{:}A_1) \rightarrow A_2}{\Gamma \vdash \mathsf{rec}\, f.v \Leftarrow A}$$

$$\frac{\Gamma \vdash a \Leftarrow A}{\Gamma \vdash a_A \Rightarrow A}$$

$$\frac{\Gamma \vdash a \Rightarrow A}{\Gamma \vdash a \Leftarrow A}$$

Inferring *proofs*

Can we infer conversion proofs, such as v in $a_{\triangleright v}$?

Coq, Agda, Cayenne, etc check types "up to β-convertibility"

$$\frac{\Gamma \vdash a : A \quad A \leadsto^* C \quad B \leadsto^* C}{\Gamma \vdash a : B}$$

Not so good for nontermination!

Inferring *proofs*

Can we infer conversion proofs, such as v in $a_{\triangleright v}$?

Coq, Agda, Cayenne, etc check types "up to β-convertibility"

$$\frac{\Gamma \vdash a : A \quad A \leadsto^{*} C \quad B \leadsto^{*} C}{\Gamma \vdash a : B}$$

Not so good for nontermination!

Our proposal: check and infer "up-to congruence closure"

$$\frac{\Gamma \vdash a \Rightarrow A \quad \Gamma \vDash |A| = |B| \quad \Gamma \vdash B \Leftarrow \mathsf{Type}}{\Gamma \vdash a \Rightarrow B}$$

$$\frac{\Gamma \vdash a \Leftarrow A \quad \Gamma \vDash |A| = |B| \quad \Gamma \vdash A \Leftarrow \mathsf{Type}}{\Gamma \vdash a \Leftarrow B}$$

(Erased) Congruence Closure

$$\frac{\Gamma \vdash a : A}{\Gamma \vDash a = a} \qquad \frac{\Gamma \vDash a = b}{\Gamma \vDash b = a} \qquad \frac{\Gamma \vDash a = b \quad \Gamma \vDash b = c}{\Gamma \vDash a = c}$$

$$\frac{x : a = b \in \Gamma}{\Gamma \vDash a = b} \qquad \frac{\overline{\Gamma \vDash a_i = b_i}^{\,i} \quad \Gamma \vdash \overline{\{a_i/x_i\}}^{\,i}\, c : A \quad \Gamma \vdash \overline{\{b_i/x_i\}}^{\,i}\, c : B}{\Gamma \vDash \overline{\{a_i/x_i\}}^{\,i}\, c = \overline{\{b_i/x_i\}}^{\,i}\, c}$$

(We will add a few more rules in the rest of the talk)

But can we implement it?

1. Algorithm to decide $\Gamma \vDash a = b$?
   Create a Union-Find structure of all subterms. Go through the given equations, adding links until nothing changes.
   - Optimized algorithm is $O(n \log n)$ [Downey-Sethi-Tarjan 1980].
2. When should the typechecker call the CC algorithm?
   Inline the conversion rules to create a syntax-directed system.

$$\frac{\begin{array}{l}\Gamma \mapsto a \Rightarrow a' : A_1 \\ \Gamma \mapsto |A_1| \Rightarrow (x : A) \to B \leadsto v_1 \\ \Gamma \mapsto v \Leftarrow A \leadsto v'\end{array}}{\Gamma \mapsto a\ v \Rightarrow (a'_{\triangleright \sim v_1 : (x:A) \to B})\ v' : \{v'/x\}B}$$

Challenges

Spoiler: dependent types makes things more difficult.

Injectivity

The algorithmic typing rule for application, first try:

$$\frac{\Gamma \mapsto a \Rightarrow A' \qquad \Gamma \mapsto |A'| = (x : A) \to B \qquad \Gamma \mapsto v \Leftarrow A}{\Gamma \mapsto a\ v \Rightarrow \{v/x\}B}$$

One worry: what if a can be assigned multiple arrow types?
E.g., suppose

$$\Gamma \models (\mathsf{Nat} \to \mathsf{Nat}) = (\mathsf{Bool} \to \mathsf{Nat})$$

Should we check v against Nat or Bool?

Injectivity for arrow domains

The problem only comes up if $\Gamma \vDash (x : A) \to B = (x : A') \to B$ but not $\Gamma \vDash A = A'$.

We avoid this by including injectivity in the core language and the CC algorithm:

$$\frac{\Gamma \vdash v : ((x : A_1) \to B_1) = ((x : A_2) \to B_2)}{\Gamma \vdash \mathsf{join}_{\mathsf{injdom}\ v} : A_1 = A_2}$$

$$\frac{\Gamma \vDash ((x : A_1) \to B_1) = ((x : A_2) \to B_2)}{\Gamma \vDash A_1 = A_2}$$

- Mildly controversial—e.g. Semantically we have $(\mathsf{Nat} \to \mathsf{Void}) = (\mathsf{Bool} \to \mathsf{Void})$.
- But we already need injectivity to prove type preservation for the core language.

Injectivity for arrow codomains?

Similarly, we are in trouble if $\Gamma \vDash (x : A) \to B' = (x : A) \to B$ but not $\Gamma \vDash \{v/x\}B = \{v/x\}B'$.

Can we use the same trick? The core language injectivity rule is type safe.

$$\frac{\Gamma \vdash v_1 : ((x : A) \to B_1) = ((x : A) \to B_2) \quad \Gamma \vdash v_2 : A}{\Gamma \vdash \mathsf{join}_{\mathsf{injrng}\ v_1\ v_2} : \{v_2/x\}B_1 = \{v_2/x\}B_2}$$

But it makes the equational theory undecidable! So we cannot add it to $\Gamma \vDash A = B$.

Injectivity for arrow codomains?

Solution: add a restriction to the *declarative* type system

$$\frac{\Gamma \vdash a \Rightarrow (x\!:\!A) \to\; B \qquad \Gamma \vdash v \Leftarrow A \qquad \Gamma \vDash \mathsf{injrng}\,(x\!:\!A) \to\; B}{\Gamma \vdash a\ v \Rightarrow \{v/x\}B}$$

where $\Gamma \vDash \mathsf{injrng}\,(x\!:\!A) \to\; B$ means, for all B',

$$\Gamma \vDash ((x\!:\!A) \to\; B) = ((x\!:\!A) \to\; B') \text{ implies } \Gamma, x : A \vDash B = B'$$

and *check* that restriction in the elaboration algorithm.

Equalities between equalities

In a dependently-typed language, we can have equations between equations.

$$(x = y) = (2 = 2)$$

We want the congruence closure relation to be stable under congruence closure. E.g.

$$h_1 : (x = y) = a, \quad h_2 : x = y \quad \vDash x = y$$

$$h_1 : (x = y) = a, \quad h_2 : a \qquad\qquad \vDash x = y$$

Equalities between equalities

In a dependently-typed language, we can have equations between equations.

$$(x = y) = (2 = 2)$$

We want the congruence closure relation to be stable under congruence closure. E.g.

$$h_1 : (x = y) = a, \quad h_2 : x = y \quad \vDash x = y$$

$$h_1 : (x = y) = a, \quad h_2 : a \qquad \vDash x = y$$

Solution: strengthen the assumption rule.

$$\frac{x : a = b \in \Gamma}{\Gamma \vDash a = b} \qquad \frac{\begin{array}{l} x : A \in \Gamma \\ \Gamma \vDash A = (a = b) \end{array}}{\Gamma \vDash a = b}$$

Typed Congruence Closure

The untyped congruence closure algorithm generates (untyped) proof terms along the way

$$p, q \quad ::= \quad x \mid \mathsf{refl} \mid p^{-1} \mid p; q \mid \mathsf{cong}_A\, p_1 \mathinner{..} p_i \mid \mathsf{inj}_i\ p$$

But not every p is a valid typed proof!

Typed Congruence Closure

The untyped congruence closure algorithm generates (untyped) proof terms along the way

$$p, q \quad ::= \quad x \mid \mathsf{refl} \mid p^{-1} \mid p; q \mid \mathsf{cong}_A\, p_1 \,..\, p_i \mid \mathsf{inj}_i\ p$$

But not every p is a valid typed proof!
Solution: simplify the proof

$$(\mathsf{cong}_A\, p_1 \,..\, p_i); (\mathsf{cong}_A\, q_1 \,..\, q_i) \quad \mapsto \quad \mathsf{cong}_A\, (p_1; q_1) \,..\, (p_1; q_i)$$

When a proof is in normal form, all intermediate terms are subterms of the wanted or the given equations, so they are well-typed.

Current Status/Future Work

Current Status

- Core language is type sound [Sjöberg et al., MSFP'12][Casinghino et al. POPL '14]
- Mostly implemented in the Zombie typechecker
- Currently working on completeness proofs for algorithmic type system and congruence closure algorithm

Future Work

- Reduction Modulo. Making join use congruence closure. E.g., if we have $h : x = \mathsf{True}$ in the context, step

$$\mathsf{if}\ x\ \mathsf{then}\ 1\ \mathsf{else}\ 2 \leadsto_{\mathsf{cbv}} 1$$

- Unification Modulo. Given two terms a and b which contain unification variables, find a substitution s such that

$$s\Gamma \vDash sa = sb$$

  This problem (*rigid E-unification*) is decidable, but NP complete.

Thanks!

Example program

```
rec minus_nn_zero : (n : Nat) → minus n n = 0.
 λ n : Nat.
   case n [n_eq] of
     Z → join [⇝ minus 0 0 = 0]
             ▷ join [minus ~n_eq ~n_eq = 0]
     S m →
       let p = minus_nn_zero m
       in
         join [⇝ minus (S m) (S m) = minus m m]
             ▷ join [minus ~n_eq ~n_eq = minus m m]
             ▷ join [minus n n = ~p]
```

Example with inference

```
rec minus_nn_zero : (n : Nat) → minus n n = 0.
 λ n.         -- infer domain type
   case n [n_eq] of
     Z → join [⇝ minus 0 0 = 0]
             -- infer conversion by n_eq
     S m →
       let p = minus_nn_zero m
       in
         join [⇝ minus (S m) (S m) = minus m m]
             -- infer conversion by n_eq
             -- and conversion by p
```

Erasure

$$
\begin{array}{lcl}
|\mathsf{Type}| & = & \mathsf{Type} \\
|x| & = & x \\
|\mathsf{rec}\ f_{A}.a| & = & \mathsf{rec}\ f.|a| \\
|(x:A) \to\ B| & = & (x:|A|) \to\ |B| \\
|\lambda x_{A}.a| & = & \lambda x.|a| \\
|a\ b| & = & |a|\ |b| \\
|a = b| & = & (|a| = |b|) \\
|\mathsf{join}_{\sigma}| & = & \mathsf{refl} \\
|a_{\triangleright b}| & = & |a|
\end{array}
$$

Desired properties of Elaboration

Lemma (Soundness)

1. *If* $\Gamma \mapsto a \Rightarrow a' : A'$ *then* $\Gamma \vdash a' : A'$
2. *If* $\Gamma \mapsto a \Leftarrow A' \rightsquigarrow a'$ *then* $\Gamma \vdash a' : A'$
3. *If* $\Gamma \mapsto A = B \rightsquigarrow v$ *then* $\Gamma \vdash v : A = B$

Lemma (Completeness)

1. *If* $\Gamma \vdash a \Rightarrow A$ *then* $\Gamma \mapsto a \Rightarrow a' : A'$
2. *If* $\Gamma \vdash a \Leftarrow A$ *then* $\Gamma \mapsto a \Leftarrow A' \rightsquigarrow a'$
3. *If* $\Gamma \vdash A = B$ *then* $\Gamma \mapsto A = B \rightsquigarrow v$