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

expressions \( a, b, c, A, B \) ::= Type | \( x \) | \( \text{rec } f_A.v \)
| (\( x:A \rightarrow B \) | \( \lambda x_A.a \) | \( a \ b \)
| \( a = b \) | join\( _\sigma \) | \( a \triangleright v \) | \ldots

coercion \( \sigma \) ::= \ldots

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

\[
\begin{align*}
\Gamma & \vdash a : A \\
\Gamma, x & : A \vdash B : Type \\
\Gamma, x & : A \vdash (x : A) \to B : Type \\
\Gamma & \vdash A : Type \\
\Gamma, x & : A \vdash B : Type \\
\Gamma & \vdash a : A \to B \\
\Gamma & \vdash b : A \\
\Gamma & \vdash v : A \\
\Gamma & \vdash a = b : Type \\
\Gamma & \vdash a \triangleright v : B
\end{align*}
\]
When are expressions equal?

- When they evaluate the same way

\[
|a| \xrightarrow{\text{cbv}} a' \quad |b| \xrightarrow{\text{cbv}} a' \quad \Gamma \vdash a = b : \text{Type}
\]

\[
\Gamma \vdash \text{join} \xrightarrow{\text{cbv}} i : a = b : a = b
\]
When are expressions equal?

- When they evaluate the same way

\[
|a| \xrightarrow{\text{cbv}}^i a' \quad |b| \xrightarrow{\text{cbv}}^j a' \quad \Gamma \vdash a = b : \text{Type}
\]

\[
\Gamma \vdash \text{join} \xrightarrow{\text{cbv}}^i j : a = b : a = b
\]

- When their subcomponents are equal (congruence)

\[
\Gamma \vdash v_j : a_j = b_j^j \quad \Gamma \vdash \{a_j/x_j\}^j c = \{b_j/x_j\}^j c : \text{Type}
\]

\[
\Gamma \vdash \text{join} \{\sim v_j/x_j\}^j c : \{a_j/x_j\}^j c = \{b_j/x_j\}^j c
\]
When are expressions equal?

- When they evaluate the same way

\[ \vdash \text{join}_{\text{cbv}}^i \ a' \rightleftharpoons \text{cbv} \ a' \quad \vdash \ a = b : \text{Type} \]

\[ \vdash \text{join}_{\text{cbv}}^i \ a'_{=b} : a = b \]

- When their subcomponents are equal (congruence)

\[ \vdash v_j : a_j = b_j^j \quad \vdash \{a_j/x_j\}^j c = \{b_j/x_j\}^j c : \text{Type} \]

\[ \vdash \text{join}_{\text{cbv}} \{\sim v_j/x_j\}^j c : \{a_j/x_j\}^j c = \{b_j/x_j\}^j c \]

- Reflexivity, symmetry and transitivity are derivable

\[ \vdash v : a = b \]

\[ \vdash \text{join}_{\text{cbv}} b_{=b} \text{join}_{\sim v = b} : b = a \]
Surface language
Inferring λ annotations: Bidirectional type system

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

\[
\begin{align*}
\Gamma \vdash a \Rightarrow A & \quad \Gamma \vdash a \Leftarrow A \\
x : A \in \Gamma & \quad \Gamma, x : A \vdash b \Leftarrow B \\
\Gamma \vdash x \Rightarrow A & \quad \Gamma \vdash \lambda x.a \Leftarrow (x : A) \to B \\
\Gamma \vdash a \Rightarrow (x : A) \to B & \quad \Gamma \vdash A \Leftarrow \text{Type} \\
\Gamma \vdash v \Leftarrow A & \quad \Gamma, f : A \vdash v \Leftarrow A \\
\Gamma \vdash a \Rightarrow \{v/x\}B & \quad A = (x : A_1) \to A_2 \\
\Gamma \vdash \text{rec } f.v \Leftarrow A & \quad \Gamma \vdash a \Leftarrow A \\
\Gamma \vdash a \Leftarrow A & \quad \Gamma \vdash a \Rightarrow A \\
\Gamma \vdash a_A \Rightarrow A & \quad \Gamma \vdash a \Leftarrow A
\end{align*}
\]
**Inferring proofs**

Can we infer conversion proofs, such as $v$ in $a \triangleright v$?

Coq, Agda, Cayenne, etc check types “up to $\beta$-convertibility”

\[
\Gamma \vdash a : A \quad A \rightsquigarrow^* C \quad B \rightsquigarrow^* C
\]

\[
\Gamma \vdash a : B
\]

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 $\beta$-convertibility”

\[
\Gamma \vdash a : A \quad A \leadsto^* C \quad B \leadsto^* C
\]

\[
\Gamma \vdash a : B
\]

Not so good for nontermination!

Our proposal: check and infer “up-to congruence closure”

\[
\Gamma \vdash a \Rightarrow A \quad \Gamma \models |A| = |B| \quad \Gamma \vdash B \Leftarrow \text{Type}
\]

\[
\Gamma \vdash a \Rightarrow B
\]

\[
\Gamma \vdash a \Leftarrow A \quad \Gamma \models |A| = |B| \quad \Gamma \vdash A \Leftarrow \text{Type}
\]

\[
\Gamma \vdash a \Leftarrow B
\]
(Erased) Congruence Closure

\[
\begin{align*}
\Gamma \vdash a : A & \quad \Gamma \vdash a = b & \quad \Gamma \vdash a = b \\
\Gamma \models a = a & \quad \Gamma \models b = a & \quad \Gamma \models a = c \\
\end{align*}
\]

\[
\begin{align*}
x : a = b \in \Gamma & \quad \Gamma \models \{a_i/x_i\}^i c : A \\
\Gamma \models a_i = b_i^i & \quad \Gamma \models \{b_i/x_i\}^i c : B \\
\Gamma \models \{a_i/x_i\}^i c = \{b_i/x_i\}^i c
\end{align*}
\]

(We will add a few more rules in the rest of the talk)
But can we implement it?

1. Algorithm to decide $\Gamma \models a = b$?
   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.

\[
\begin{align*}
\Gamma \vdash a & \Rightarrow a' : A_1 \\
\Gamma \vdash |A_1| & \Rightarrow (x : A) \rightarrow B \rightsquigarrow v_1 \\
\Gamma \vdash v & \Leftarrow A \rightsquigarrow v' \\
\hline
\Gamma \vdash a \; v & \Rightarrow (a' \; \triangleright_{v_1} (x : A) \rightarrow B) \; v' : \{v'/x\}B
\end{align*}
\]
Challenges

Spoiler: dependent types makes things more difficult.
Injectivity

The algorithmic typing rule for application, first try:

\[
\begin{align*}
\Gamma \vDash a \Rightarrow A' \\
\Gamma \vDash |A'| = (x : A) \rightarrow B \\
\Gamma \vDash v \Leftarrow A \\
\hline
\Gamma \vDash a \; v \Rightarrow \{v/x\}B
\end{align*}
\]

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

\[
\Gamma \vdash (\text{Nat} \rightarrow \text{Nat}) = (\text{Bool} \rightarrow \text{Nat})
\]

Should we check \(v\) against \text{Nat} or \text{Bool}?
Injectivity for arrow domains

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

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

\[
\Gamma \vdash v : ((x : A_1) \to B_1) = ((x : A_2) \to B_2) \\
\Gamma \vdash \text{join}_{\text{injdom}} v : A_1 = A_2
\]

\[
\Gamma \models ((x : A_1) \to B_1) = ((x : A_2) \to B_2) \\
\Gamma \models A_1 = A_2
\]

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

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

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

$$
\Gamma \vdash v_1 : ((x : A) \rightarrow B_1) = ((x : A) \rightarrow B_2) \quad \Gamma \vdash v_2 : A
$$

$$
\Gamma \vdash \text{join}_{\text{injrng}} v_1 v_2 : \{ v_2/x \} B_1 = \{ v_2/x \} B_2
$$

But it makes the equational theory undecidable! So we cannot add it to $\Gamma \models A = B$. 
 Injectivity for arrow codomains?

Solution: add a restriction to the declarative type system

\[ \Gamma \vdash a \Rightarrow (x:A) \rightarrow B \]
\[ \Gamma \vdash v \Leftarrow A \]
\[ \Gamma \vdash \text{injrng} (x:A) \rightarrow B \]
\[ \Gamma \vdash a \, v \Rightarrow \{v/x\}B \]

where \( \Gamma \vdash \text{injrng} (x:A) \rightarrow B \) means, for all \( B' \),

\[ \Gamma \vdash ((x:A) \rightarrow B) = ((x:A) \rightarrow B') \] 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.

\[h_1 : (x = y) = a, \quad h_2 : x = y \quad \vdash x = y\]

\[h_1 : (x = y) = a, \quad h_2 : a \quad \vdash x = y\]
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.

\[h_1 : (x = y) = a, \quad h_2 : x = y \vdash x = y\]

\[h_1 : (x = y) = a, \quad h_2 : a \vdash x = y\]

Solution: strengthen the assumption rule.

\[
\begin{array}{c}
x : a = b \in \Gamma \\
\hline
\Gamma \vdash a = b
\end{array} \quad \begin{array}{c}
x : A \in \Gamma \\
\Gamma \vdash A = (a = b)
\end{array}
\]
Typed Congruence Closure

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

\[ p, q ::= x \mid \text{refl} \mid p^{-1} \mid p; q \mid \text{cong}_A p_1 .. p_i \mid \text{inj}_i p \]

But not every \( p \) is a valid typed proof!
Typed Congruence Closure

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

\[ p, q ::= x \mid \text{refl} \mid p^{-1} \mid p; q \mid \text{cong}_A p_1 \ldots p_i \mid \text{inj}_i p \]

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

\[ (\text{cong}_A p_1 \ldots p_i); (\text{cong}_A q_1 \ldots q_i) \rightarrow \text{cong}_A (p_1; q_1) \ldots (p_1; q_i) \]

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 $h : x = \text{True}$ in the context, step

$$\text{if } x \text{ then } 1 \text{ else } 2 \rightsquigarrow_{\text{cbv}} 1$$

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

$$s\Gamma \models sa = sb$$

This problem (rigid $E$-unification) is decidable, but NP complete.
Thanks!
Example program

```ocaml
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 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
```
| Type | = | Type  
| x   | = | x      
| rec $f_A \cdot a$ | = | rec $f \cdot a$  
| $(x : A) \rightarrow B$ | = | $(x : |A|) \rightarrow |B|$  
| $\lambda x_A \cdot a$ | = | $\lambda x \cdot a$  
| $a \; b$ | = | $|a| \; |b|$  
| $a = b$ | = | $(|a| = |b|)$  
| join$_{\sigma}$ | = | refl  
| $a \triangleright b$ | = | $|a|$  

Erasure
Desired properties of Elaboration

**Lemma (Soundness)**

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

**Lemma (Completeness)**

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