Towards Unification for Dependent Types

Ningning Xie, Bruno C. d. S. Oliveira

The University of Hong Kong

June 2017

Outline

Outline

Motivation

- Developments on type unification techniques for sophisticated dependent type systems.
  - Features: higher-order, polymorphism, subtyping, etc.
  - powerful, but complicated, complex, and hard to reason.

META-SAME-SAME
$$\frac{\Sigma;\Gamma \vdash \overline{t} \approx_{\equiv} \overline{u} \triangleright \Sigma'}{\Sigma;\Gamma \vdash ?x[\sigma]\ \overline{t} \approx_{\mathcal{R}} ?x[\sigma]\ \overline{u} \triangleright \Sigma'}$$

META-SAME
$$\frac{?x : T[\Psi_1] \in \Sigma \qquad \Psi_1 \vdash \sigma \cap \sigma' \triangleright \Psi_2 \qquad \cdot \vdash \mathsf{sanitize}(\Psi_2) \triangleright \Psi_3 \qquad \mathsf{FV}(T) \subseteq \Psi_3 \qquad \Sigma \cup \{?y : T[\Psi_3], ?x := ?y[\widehat{\Psi_3}]\};\Gamma \vdash \overline{t} \approx_{\equiv} \overline{u} \triangleright \Sigma'}{\Sigma;\Gamma \vdash ?x[\sigma]\ \overline{t} \approx_{\mathcal{R}} ?x[\sigma']\ \overline{u} \triangleright \Sigma'}$$

META-INSTR
$$\frac{\begin{array}{c} ?x : T[\Psi] \in \Sigma_0 \qquad t', \xi_1 = \mathsf{remove\_tail}(t; \xi') \qquad t' \downarrow^{w}_{\beta} t'' \qquad \Sigma_0 \vdash \mathsf{prune}(?x; \xi, \xi_1; t'') \triangleright \Sigma_1 \qquad \Sigma_1;\Gamma \vdash \xi_1 : \overline{U} \\ t''' = \lambda y : U\{\xi, \xi_1/\hat{\Psi}, \overline{y}\}^{-1}.\ \Sigma_1(t'')\{\xi, \xi_1/\hat{\Psi}, \overline{y}\}^{-1} \qquad \Sigma_1;\Psi \vdash t''' : T' \qquad \Sigma_1;\Psi \vdash T' \approx_{\leq} T \triangleright \Sigma_2 \qquad ?x \notin \mathsf{FMV}(t''') \end{array}}{\Sigma_0;\Gamma \vdash t \approx_{\mathcal{R}} ?x[\xi]\ \xi' \triangleright \Sigma_2 \cup \{?x := t'''\}}$$

META-FOR
$$\frac{?x : T[\Psi] \in \Sigma_0 \qquad 0 < n \qquad \Sigma_0;\Gamma \vdash u\ \overline{u'_m} \approx_{\equiv} ?x[\sigma] \triangleright \Sigma_1 \qquad \Sigma_1;\Gamma \vdash \overline{u''_n} \approx_{\equiv} \overline{t_n} \triangleright \Sigma_2}{\Sigma_0;\Gamma \vdash u\ \overline{u'_m u''_n} \approx_{\mathcal{R}} ?x[\sigma]\ \overline{t_n} \triangleright \Sigma_2}$$

[1]

[1] Ziliani, Beta, and Matthieu Sozeau. "A unification algorithm for Coq featuring universe polymorphism and overloading." ACM SIGPLAN Notices. Vol. 50. No. 9. ACM, 2015.

Motivation

- Developments on type unification techniques for sophisticated dependent type systems.
  - Features: higher-order, polymorphism, subtyping, etc.
  - powerful, but complicated, complex, and hard to reason.
- Developments on dependent type systems that give programmers more control.
  - Manage type-level computations using explicit casts. [1] [2] [3] [4]
  - Decidable type checking based on alpha-equality.
  - Easy to combine recursive types.

$$\frac{\Gamma \vdash e : \tau_2 \qquad \Gamma \vdash \tau_1 : \star \qquad \tau_1 \longrightarrow \tau_2}{\Gamma \vdash \mathsf{cast}_{\uparrow} [\tau_1]\ e : \tau_1} \text{T\_CastUp} \qquad \frac{\Gamma \vdash e : \tau_1 \qquad \tau_1 \longrightarrow \tau_2}{\Gamma \vdash \mathsf{cast}_{\downarrow}\ e : \tau_2} \text{T\_CastDown}$$

[1] Yang, Yanpeng, Xuan Bi, and Bruno C. D. S. Oliveira. "Unified Syntax with Iso-types." Asian Symposium on Programming Languages and Systems. Springer International Publishing, 2016.

[2] van Doorn, Floris, Herman Geuvers, and Freek Wiedijk. "Explicit convertibility proofs in pure type systems." Proceedings of the Eighth ACM SIGPLAN international workshop on Logical frameworks & meta-languages: theory & practice. ACM, 2013.

[3] Kimmell, Garrin, et al. "Equational reasoning about programs with general recursion and call-by-value semantics." Proceedings of the sixth workshop on Programming languages meets program verification. ACM, 2012.

[4] Sjberg, Vilhelm, and Stephanie Weirich. "Programming up to congruence." ACM SIGPLAN Notices. Vol. 50. No. 1. ACM,

Motivation

- Developments on type unification techniques for sophisticated dependent type systems.
    - Features: higher-order, polymorphism, subtyping, etc.
    - powerful, but complicated, complex, and hard to reason.
- Developments on dependent type systems that give programmers more control.
    - Manage type-level computations using explicit casts.
    - Decidable type checking based on alpha-equality.
    - Easy to combine recursive types.
- Question: can we get rid of the complication of the algorithms in those systems?

Goals

Our goal is to

- present a simple and complete unification algorithm for first-order dependent type systems with alpha-equality based type checking
- fill the gap between delicate unification algorithms for simple types and sophisticated unification algorithms for dependent types.

We do *not* intend to

- solve more problems than existing unification algorithms.
- serve for beta-equality based dependent type systems.

Contributions

- Strategy: *type sanitization* that resolves the dependency between types.
- Algorithm: an alpha-equality based unification algorithm for first-order dependent types.
- Extension: subtyping in implicit polymorphism.
- Meta-theory Study: undergoing.

Background: Dependent Types

- Types depends on terms.
- Vector of integers
  - definition without dependent types: $data\ Vect = Nil \mid Cons\ Int\ Vect$

Background: Dependent Types

- Types depends on terms.
- Vector of integers
  - definition without dependent types: $data\ Vect = Nil \mid Cons\ Int\ Vect$
    - one definition that could cause run-time error
      $head :: Vect \rightarrow Int$

Background: Dependent Types

- Types depends on terms.
- Vector of integers
  - definition without dependent types: $data\ Vect = Nil \mid Cons\ Int\ Vect$
    - one definition that could cause run-time error
      $head :: Vect \rightarrow Int$
    - make it total
      $head :: Vect \rightarrow Maybe\ Int$

Background: Dependent Types

- Types depends on terms.
- Vector of integers
  - definition without dependent types: $\mathit{data}\ \mathit{Vect} = \mathit{Nil} \mid \mathit{Cons}\ \mathit{Int}\ \mathit{Vect}$
    - one definition that could cause run-time error
      $\mathit{head} :: \mathit{Vect} \to \mathit{Int}$
    - make it total
      $\mathit{head} :: \mathit{Vect} \to \mathit{Maybe}\ \mathit{Int}$
  - definition with dependent type: sized Vector

$$
\begin{array}{l}
\textbf{data}\ \mathit{Vect} :: \mathit{Nat} \to \mathit{Type}\ = \\
\quad \mid \mathit{Nil} :: \mathit{Vect}\ Z \\
\quad \mid \mathit{Cons} :: \mathit{Int} \to \mathit{Vect}\ k \to \mathit{Vect}\ (S\ k)
\end{array}
$$

Background: Dependent Types

- Types depends on terms.
- Vector of integers
  - definition without dependent types: *data Vect = Nil | Cons Int Vect*
    - one definition that could cause run-time error
      head :: Vect → Int
    - make it total
      head :: Vect → Maybe Int
  - definition with dependent type: sized Vector

$$\begin{aligned}&\textbf{data}\ \mathit{Vect} :: \mathit{Nat} \to \mathit{Type}\ =\\ &\quad \mid \mathit{Nil} :: \mathit{Vect}\ Z\\ &\quad \mid \mathit{Cons} :: \mathit{Int} \to \mathit{Vect}\ k \to \mathit{Vect}\ (S\ k)\end{aligned}$$

$$\mathit{head} :: \mathit{Vect}\ (S\ k) \to \mathit{Int}$$

Background: Unification Problem

Unification

Given two terms containing some unification variables, find the substitution which makes two terms equal.

Background: Unification Problem

Unification

Given two terms containing some unification variables, find the substitution which makes two terms equal.

- $\widehat{\alpha} \to Int$
- $Bool \to Int$

Background: Unification Problem

Unification

Given two terms containing some unification variables, find the substitution which makes two terms equal.

- $\widehat{\alpha} \to Int$
- $Bool \to Int$

Solution: $\widehat{\alpha} = Bool$.

Outline

Language

- Unified syntax based on λC

Syntax

Type $\sigma, \tau ::= \widehat{\alpha} \mid e$

Expr $e ::= x \mid \star \mid e_1\ e_2 \mid \lambda x : \sigma.\ e \mid \Pi x : \sigma_1.\ \sigma_2$

- $\lambda x.\ e \equiv \lambda x : \widehat{\alpha}.\ e$
- Example: $(\lambda x : \star.\ \lambda y : x.\ y) :: \Pi x : \star.\ \Pi y : x.\ x$
- $A \to B$ for $\Pi x : A.\ B$ if x does not appear in B.

Unification Algorithm

Key ideas:

- ordered typing context [1]:

Algorithmic typing context

Contexts $\Gamma, \Theta, \Delta ::= \varnothing \mid \Gamma, x : \sigma \mid \Gamma, \widehat{\alpha} \mid \Gamma, \widehat{\alpha} = \tau$

scope constraint

- $\lambda x : \widehat{\alpha}.\ \lambda y : \widehat{\beta}.\ y$
- $\widehat{\alpha} = y$ invalid
- $\widehat{\beta} = x$ valid

[1] Dunfield, Joshua, and Neelakantan R. Krishnaswami. "Complete and easy bidirectional typechecking for higher-rank polymorphism." ACM SIGPLAN Notices. Vol. 48. No. 9. ACM, 2013.

Unification Algorithm

Key ideas:

- ordered typing context [1]:

Algorithmic typing context

Contexts $\Gamma, \Theta, \Delta ::= \varnothing \mid \Gamma, x : \sigma \mid \Gamma, \widehat{\alpha} \mid \Gamma, \widehat{\alpha} = \tau$

scope constraint

- $\lambda x : \widehat{\alpha}.\ \lambda y : \widehat{\beta}.\ y$
- $\widehat{\alpha} = y$ invalid
- $\widehat{\beta} = x$ valid

- judgment: $\Gamma \vdash \tau_1 \simeq \tau_2 \dashv \Theta$

[1] Dunfield, Joshua, and Neelakantan R. Krishnaswami. "Complete and easy bidirectional typechecking for higher-rank polymorphism." ACM SIGPLAN Notices. Vol. 48. No. 9. ACM, 2013.

Unification Algorithm

Key ideas:

- ordered typing context [1]:

Algorithmic typing context

Contexts $\Gamma, \Theta, \Delta ::= \varnothing \mid \Gamma, x : \sigma \mid \Gamma, \widehat{\alpha} \mid \Gamma, \widehat{\alpha} = \tau$

scope constraint

- $\lambda x : \widehat{\alpha}.\ \lambda y : \widehat{\beta}.\ y$
- $\widehat{\alpha} = y$ invalid
- $\widehat{\beta} = x$ valid

- judgment: $\Gamma \vdash \tau_1 \simeq \tau_2 \dashv \Theta$
- invariant: inputs are already fully substituted under current context.
  - $\widehat{\alpha} = \mathit{Int} \vdash \widehat{\alpha} \simeq \mathit{Bool}$ invalid
  - $\widehat{\alpha} = \mathit{Int} \vdash \mathit{Int} \simeq \mathit{Bool}$ valid

[1] Dunfield, Joshua, and Neelakantan R. Krishnaswami. "Complete and easy bidirectional typechecking for higher-rank polymorphism." ACM SIGPLAN Notices. Vol. 48. No. 9. ACM, 2013.

Problem

The case when we have a unification variable on one side:

- $\Gamma, \widehat{\alpha}, \Delta \vdash \widehat{\alpha} \simeq \tau$

[2] Dunfield, Joshua, and Neelakantan R. Krishnaswami. "Complete and easy bidirectional typechecking for higher-rank polymorphism." ACM SIGPLAN Notices. Vol. 48. No. 9. ACM, 2013.

Problem

The case when we have a unification variable on one side:

- $\Gamma, \widehat{\alpha}, \Delta \vdash \widehat{\alpha} \simeq \tau$
- what we will do if τ is a function? $\Gamma, \widehat{\alpha}, \Delta \vdash \widehat{\alpha} \simeq A \rightarrow B$

[2]Dunfield, Joshua, and Neelakantan R. Krishnaswami. "Complete and easy bidirectional typechecking for higher-rank polymorphism." ACM SIGPLAN Notices. Vol. 48. No. 9. ACM, 2013.

Problem

The case when we have a unification variable on one side:

- $\Gamma, \widehat{\alpha}, \Delta \vdash \widehat{\alpha} \simeq \tau$
- what we will do if τ is a function? $\Gamma, \widehat{\alpha}, \Delta \vdash \widehat{\alpha} \simeq A \to B$
  - try directly scope constraint? No.

[2] Dunfield, Joshua, and Neelakantan R. Krishnaswami. "Complete and easy bidirectional typechecking for higher-rank polymorphism." ACM SIGPLAN Notices. Vol. 48. No. 9. ACM, 2013.

Problem

The case when we have a unification variable on one side:

- $\Gamma, \widehat{\alpha}, \Delta \vdash \widehat{\alpha} \simeq \tau$
- what we will do if τ is a function? $\Gamma, \widehat{\alpha}, \Delta \vdash \widehat{\alpha} \simeq A \to B$
    - try directly scope constraint? No.
    - $\widehat{\alpha}, \widehat{\beta} \vdash \widehat{\alpha} \simeq \mathit{Int} \to \widehat{\beta}$

[2] Dunfield, Joshua, and Neelakantan R. Krishnaswami. "Complete and easy bidirectional typechecking for higher-rank polymorphism." ACM SIGPLAN Notices. Vol. 48. No. 9. ACM, 2013.

Problem

The case when we have a unification variable on one side:

- $\Gamma, \widehat{\alpha}, \Delta \vdash \widehat{\alpha} \simeq \tau$
- what we will do if τ is a function? $\Gamma, \widehat{\alpha}, \Delta \vdash \widehat{\alpha} \simeq A \rightarrow B$
    - try directly scope constraint? No.
    - $\widehat{\alpha}, \widehat{\beta} \vdash \widehat{\alpha} \simeq Int \rightarrow \widehat{\beta}$
    - $\widehat{\alpha}_1, \widehat{\alpha} = Int \rightarrow \widehat{\alpha}_1, \widehat{\beta} = \widehat{\alpha}_1$

[2] Dunfield, Joshua, and Neelakantan R. Krishnaswami. "Complete and easy bidirectional typechecking for higher-rank polymorphism." ACM SIGPLAN Notices. Vol. 48. No. 9. ACM, 2013.

Problem

The case when we have a unification variable on one side:

- $\Gamma, \widehat{\alpha}, \Delta \vdash \widehat{\alpha} \simeq \tau$
- what we will do if τ is a function? $\Gamma, \widehat{\alpha}, \Delta \vdash \widehat{\alpha} \simeq A \rightarrow B$
  - try directly scope constraint? No.
  - $\widehat{\alpha}, \widehat{\beta} \vdash \widehat{\alpha} \simeq Int \rightarrow \widehat{\beta}$
  - $\widehat{\alpha}_1, \widehat{\alpha} = Int \rightarrow \widehat{\alpha}_1, \widehat{\beta} = \widehat{\alpha}_1$
  - unification variables need special treatments in scope constraint!

[2] Dunfield, Joshua, and Neelakantan R. Krishnaswami. "Complete and easy bidirectional typechecking for higher-rank polymorphism." ACM SIGPLAN Notices. Vol. 48. No. 9. ACM, 2013.

Problem

The case when we have a unification variable on one side:

- $\Gamma, \widehat{\alpha}, \Delta \vdash \widehat{\alpha} \simeq \tau$
- what we will do if τ is a function? $\Gamma, \widehat{\alpha}, \Delta \vdash \widehat{\alpha} \simeq A \to B$
  - try directly scope constraint? No.
  - $\widehat{\alpha}, \widehat{\beta} \vdash \widehat{\alpha} \simeq \mathit{Int} \to \widehat{\beta}$
  - $\widehat{\alpha}_1, \widehat{\alpha} = \mathit{Int} \to \widehat{\alpha}_1, \widehat{\beta} = \widehat{\alpha}_1$
  - unification variables need special treatments in scope constraint!
- In Dunfield and Krishnaswami 2013 [2]:

[2] Dunfield, Joshua, and Neelakantan R. Krishnaswami. "Complete and easy bidirectional typechecking for higher-rank polymorphism." ACM SIGPLAN Notices. Vol. 48. No. 9. ACM, 2013.

Problem

The case when we have a unification variable on one side:

- $\Gamma, \widehat{\alpha}, \Delta \vdash \widehat{\alpha} \simeq \tau$
- what we will do if τ is a function? $\Gamma, \widehat{\alpha}, \Delta \vdash \widehat{\alpha} \simeq A \rightarrow B$
  - try directly scope constraint? No.
  - $\widehat{\alpha}, \widehat{\beta} \vdash \widehat{\alpha} \simeq Int \rightarrow \widehat{\beta}$
  - $\widehat{\alpha}_1, \widehat{\alpha} = Int \rightarrow \widehat{\alpha}_1, \widehat{\beta} = \widehat{\alpha}_1$
  - unification variables need special treatments in scope constraint!
- In Dunfield and Krishnaswami 2013 [2]:
  - solve $\widehat{\alpha} = \widehat{\alpha}_1 \rightarrow \widehat{\alpha}_2$.
  - unify $\Gamma_1, \widehat{\alpha}_1, \widehat{\alpha}_2, \widehat{\alpha} = \widehat{\alpha}_1 \rightarrow \widehat{\alpha}_2, \Gamma_2 \vdash \widehat{\alpha}_1 \simeq A$
    $\Gamma_1, \widehat{\alpha}_1, \widehat{\alpha}_2, \widehat{\alpha} = \widehat{\alpha}_1 \rightarrow \widehat{\alpha}_2, \Gamma_2 \vdash \widehat{\alpha}_2 \simeq B$

[2] Dunfield, Joshua, and Neelakantan R. Krishnaswami. "Complete and easy bidirectional typechecking for higher-rank polymorphism." ACM SIGPLAN Notices. Vol. 48. No. 9. ACM, 2013.

Problem

The case when we have a unification variable on one side:

- $\Gamma, \widehat{\alpha}, \Delta \vdash \widehat{\alpha} \simeq \tau$
- what we will do if τ is a function? $\Gamma, \widehat{\alpha}, \Delta \vdash \widehat{\alpha} \simeq A \to B$
  - try directly scope constraint? No.
  - $\widehat{\alpha}, \widehat{\beta} \vdash \widehat{\alpha} \simeq \mathit{Int} \to \widehat{\beta}$
  - $\widehat{\alpha}_1, \widehat{\alpha} = \mathit{Int} \to \widehat{\alpha}_1, \widehat{\beta} = \widehat{\alpha}_1$
  - unification variables need special treatments in scope constraint!
- In Dunfield and Krishnaswami 2013 [2]:
  - solve $\widehat{\alpha} = \widehat{\alpha}_1 \to \widehat{\alpha}_2$.
  - unify $\Gamma_1, \widehat{\alpha}_1, \widehat{\alpha}_2, \widehat{\alpha} = \widehat{\alpha}_1 \to \widehat{\alpha}_2, \Gamma_2 \vdash \widehat{\alpha}_1 \simeq A$
    $\Gamma_1, \widehat{\alpha}_1, \widehat{\alpha}_2, \widehat{\alpha} = \widehat{\alpha}_1 \to \widehat{\alpha}_2, \Gamma_2 \vdash \widehat{\alpha}_2 \simeq B$
- Then $\Gamma_1, \widehat{\alpha}, \Gamma_2 \vdash \widehat{\alpha} \simeq \Pi x : \star.\ x$?

[2] Dunfield, Joshua, and Neelakantan R. Krishnaswami. "Complete and easy bidirectional typechecking for higher-rank polymorphism." ACM SIGPLAN Notices. Vol. 48. No. 9. ACM, 2013.

Problem

The case when we have a unification variable on one side:

- $\Gamma, \widehat{\alpha}, \Delta \vdash \widehat{\alpha} \simeq \tau$
- what we will do if τ is a function? $\Gamma, \widehat{\alpha}, \Delta \vdash \widehat{\alpha} \simeq A \rightarrow B$
  - try directly scope constraint? No.
  - $\widehat{\alpha}, \widehat{\beta} \vdash \widehat{\alpha} \simeq Int \rightarrow \widehat{\beta}$
  - $\widehat{\alpha}_1, \widehat{\alpha} = Int \rightarrow \widehat{\alpha}_1, \widehat{\beta} = \widehat{\alpha}_1$
  - unification variables need special treatments in scope constraint!
- In Dunfield and Krishnaswami 2013 [2]:
  - solve $\widehat{\alpha} = \widehat{\alpha}_1 \rightarrow \widehat{\alpha}_2$.
  - unify $\Gamma_1, \widehat{\alpha}_1, \widehat{\alpha}_2, \widehat{\alpha} = \widehat{\alpha}_1 \rightarrow \widehat{\alpha}_2, \Gamma_2 \vdash \widehat{\alpha}_1 \simeq A$
    $\Gamma_1, \widehat{\alpha}_1, \widehat{\alpha}_2, \widehat{\alpha} = \widehat{\alpha}_1 \rightarrow \widehat{\alpha}_2, \Gamma_2 \vdash \widehat{\alpha}_2 \simeq B$
- Then $\Gamma_1, \widehat{\alpha}, \Gamma_2 \vdash \widehat{\alpha} \simeq \Pi x : \star.\ x$?
  - $\widehat{\alpha}_2 = x$

[2] Dunfield, Joshua, and Neelakantan R. Krishnaswami. "Complete and easy bidirectional typechecking for higher-rank polymorphism." ACM SIGPLAN Notices. Vol. 48. No. 9. ACM, 2013.

Problem

The case when we have a unification variable on one side:

- $\Gamma, \widehat{\alpha}, \Delta \vdash \widehat{\alpha} \simeq \tau$
- what we will do if τ is a function? $\Gamma, \widehat{\alpha}, \Delta \vdash \widehat{\alpha} \simeq A \rightarrow B$
  - try directly scope constraint? No.
  - $\widehat{\alpha}, \widehat{\beta} \vdash \widehat{\alpha} \simeq Int \rightarrow \widehat{\beta}$
  - $\widehat{\alpha}_1, \widehat{\alpha} = Int \rightarrow \widehat{\alpha}_1, \widehat{\beta} = \widehat{\alpha}_1$
  - unification variables need special treatments in scope constraint!
- In Dunfield and Krishnaswami 2013 [2]:
  - solve $\widehat{\alpha} = \widehat{\alpha}_1 \rightarrow \widehat{\alpha}_2$.
  - unify $\Gamma_1, \widehat{\alpha}_1, \widehat{\alpha}_2, \widehat{\alpha} = \widehat{\alpha}_1 \rightarrow \widehat{\alpha}_2, \Gamma_2 \vdash \widehat{\alpha}_1 \simeq A$
    $\Gamma_1, \widehat{\alpha}_1, \widehat{\alpha}_2, \widehat{\alpha} = \widehat{\alpha}_1 \rightarrow \widehat{\alpha}_2, \Gamma_2 \vdash \widehat{\alpha}_2 \simeq B$
- Then $\Gamma_1, \widehat{\alpha}, \Gamma_2 \vdash \widehat{\alpha} \simeq \Pi x : \star .\ x$?
  - $\widehat{\alpha}_2 = x$
  - $\Gamma_1, \widehat{\alpha}_1, x, \widehat{\alpha}_2, \widehat{\alpha} = \widehat{\alpha}_1 \rightarrow \widehat{\alpha}_2, \Gamma_2$.

[2] Dunfield, Joshua, and Neelakantan R. Krishnaswami. "Complete and easy bidirectional typechecking for higher-rank polymorphism." ACM SIGPLAN Notices. Vol. 48. No. 9. ACM, 2013.

Problem

The case when we have a unification variable on one side:

- $\Gamma, \widehat{\alpha}, \Delta \vdash \widehat{\alpha} \simeq \tau$
- what we will do if τ is a function? $\Gamma, \widehat{\alpha}, \Delta \vdash \widehat{\alpha} \simeq A \to B$
  - try directly scope constraint? No.
  - $\widehat{\alpha}, \widehat{\beta} \vdash \widehat{\alpha} \simeq Int \to \widehat{\beta}$
  - $\widehat{\alpha}_1, \widehat{\alpha} = Int \to \widehat{\alpha}_1, \widehat{\beta} = \widehat{\alpha}_1$
  - unification variables need special treatments in scope constraint!
- In Dunfield and Krishnaswami 2013 [2]:
  - solve $\widehat{\alpha} = \widehat{\alpha}_1 \to \widehat{\alpha}_2$.
  - unify $\Gamma_1, \widehat{\alpha}_1, \widehat{\alpha}_2, \widehat{\alpha} = \widehat{\alpha}_1 \to \widehat{\alpha}_2, \Gamma_2 \vdash \widehat{\alpha}_1 \simeq A$
    $\Gamma_1, \widehat{\alpha}_1, \widehat{\alpha}_2, \widehat{\alpha} = \widehat{\alpha}_1 \to \widehat{\alpha}_2, \Gamma_2 \vdash \widehat{\alpha}_2 \simeq B$
- Then $\Gamma_1, \widehat{\alpha}, \Gamma_2 \vdash \widehat{\alpha} \simeq \Pi x : \star .\ x$?
  - $\widehat{\alpha}_2 = x$
  - $\Gamma_1, \widehat{\alpha}_1, x, \widehat{\alpha}_2, \widehat{\alpha} = \widehat{\alpha}_1 \to \widehat{\alpha}_2, \Gamma_2$.

[2] Dunfield, Joshua, and Neelakantan R. Krishnaswami. "Complete and easy bidirectional typechecking for higher-rank polymorphism." ACM SIGPLAN Notices. Vol. 48. No. 9. ACM, 2013.

Problem

The case when we have a unification variable on one side:

- $\Gamma, \widehat{\alpha}, \Delta \vdash \widehat{\alpha} \simeq \tau$
- what we will do if τ is a function? $\Gamma, \widehat{\alpha}, \Delta \vdash \widehat{\alpha} \simeq A \to B$
  - try directly scope constraint? No.
  - $\widehat{\alpha}, \widehat{\beta} \vdash \widehat{\alpha} \simeq Int \to \widehat{\beta}$
  - $\widehat{\alpha}_1, \widehat{\alpha} = Int \to \widehat{\alpha}_1, \widehat{\beta} = \widehat{\alpha}_1$
  - unification variables need special treatments in scope constraint!
- In Dunfield and Krishnaswami 2013 [2]:
  - solve $\widehat{\alpha} = \widehat{\alpha}_1 \to \widehat{\alpha}_2$.
  - unify $\Gamma_1, \widehat{\alpha}_1, \widehat{\alpha}_2, \widehat{\alpha} = \widehat{\alpha}_1 \to \widehat{\alpha}_2, \Gamma_2 \vdash \widehat{\alpha}_1 \simeq A$
    $\Gamma_1, \widehat{\alpha}_1, \widehat{\alpha}_2, \widehat{\alpha} = \widehat{\alpha}_1 \to \widehat{\alpha}_2, \Gamma_2 \vdash \widehat{\alpha}_2 \simeq B$
- Then $\Gamma_1, \widehat{\alpha}, \Gamma_2 \vdash \widehat{\alpha} \simeq \Pi x : \star .\ x$?
  - $\widehat{\alpha}_2 = x$
  - $\Gamma_1, \widehat{\alpha}_1, x, \widehat{\alpha}_2, \widehat{\alpha} = \widehat{\alpha}_1 \to \widehat{\alpha}_2, \Gamma_2$. No.

[2] Dunfield, Joshua, and Neelakantan R. Krishnaswami. "Complete and easy bidirectional typechecking for higher-rank polymorphism." ACM SIGPLAN Notices. Vol. 48. No. 9. ACM, 2013.

Problem

- can not use scope constraint directly because of unification variables
  - $\widehat{\alpha}, \widehat{\beta} \vdash \widehat{\alpha} \simeq \mathit{Int} \rightarrow \widehat{\beta}$
  - $\widehat{\alpha}_1, \widehat{\alpha} = \mathit{Int} \rightarrow \widehat{\alpha}_1, \widehat{\beta} = \widehat{\alpha}_1$
- cannot destruct a Pi type because of the type dependency
  - $\Gamma_1, \widehat{\alpha}, \Gamma_2 \vdash \widehat{\alpha} \simeq \Pi x : \star.\ x$

Problem

- can not use scope constraint directly because of unification variables
  - $\widehat{\alpha}, \widehat{\beta} \vdash \widehat{\alpha} \simeq \mathit{Int} \to \widehat{\beta}$
  - $\widehat{\alpha}_1, \widehat{\alpha} = \mathit{Int} \to \widehat{\alpha}_1, \widehat{\beta} = \widehat{\alpha}_1$
- cannot destruct a Pi type because of the type dependency
  - $\Gamma_1, \widehat{\alpha}, \Gamma_2 \vdash \widehat{\alpha} \simeq \Pi x : \star.\ x$
- observation: we can always solve it by a fresh unification variable that satisfies the scope constraint.

Problem

- can not use scope constraint directly because of unification variables
  - $\widehat{\alpha}, \widehat{\beta} \vdash \widehat{\alpha} \simeq \mathit{Int} \to \widehat{\beta}$
  - $\widehat{\alpha}_1, \widehat{\alpha} = \mathit{Int} \to \widehat{\alpha}_1, \widehat{\beta} = \widehat{\alpha}_1$
- cannot destruct a Pi type because of the type dependency
  - $\Gamma_1, \widehat{\alpha}, \Gamma_2 \vdash \widehat{\alpha} \simeq \Pi x : \star.\ x$
- observation: we can always solve it by a fresh unification variable that satisfies the scope constraint.
- Our solution: for unification problem $\Gamma, \widehat{\alpha}, \Delta \vdash \widehat{\alpha} \simeq \tau$, we sanitize the unification variables in τ before we check the scope constraint.

Strategy

Type Sanitization

Given $\widehat{\alpha}, \tau$, solve unification variables in τ out of scope of $\widehat{\alpha}$ by fresh unification variables that in that scope of $\widehat{\alpha}$.

Strategy

Type Sanitization

Given $\widehat{\alpha}, \tau$, solve unification variables in τ out of scope of $\widehat{\alpha}$ by fresh unification variables that in that scope of $\widehat{\alpha}$.

Example

Strategy

Type Sanitization

Given $\widehat{\alpha}, \tau$, solve unification variables in τ out of scope of $\widehat{\alpha}$ by fresh unification variables that in that scope of $\widehat{\alpha}$.

Example

- $\widehat{\alpha}, \widehat{\beta} \vdash \widehat{\alpha} \simeq \mathit{Int} \to \widehat{\beta}$
  - type sanitization: $\widehat{\alpha}, \widehat{\beta} \vdash \mathit{Int} \to \widehat{\beta} \longmapsto \mathit{Int} \to \widehat{\alpha}_1 \dashv \widehat{\alpha}_1, \widehat{\alpha}, \widehat{\beta} = \widehat{\alpha}_1$
  - after scope constraint: $\widehat{\alpha}_1, \widehat{\alpha} = \mathit{Int} \to \widehat{\alpha}_1, \widehat{\beta} = \widehat{\alpha}_1$

Type Sanitization

Given $\widehat{\alpha}, \tau$, solve unification variables in τ out of scope of $\widehat{\alpha}$ by fresh unification variables that in that scope of $\widehat{\alpha}$.

Example

- $\widehat{\alpha}, \widehat{\beta} \vdash \widehat{\alpha} \simeq \mathit{Int} \to \widehat{\beta}$
  - type sanitization: $\widehat{\alpha}, \widehat{\beta} \vdash \mathit{Int} \to \widehat{\beta} \longmapsto \mathit{Int} \to \widehat{\alpha}_1 \dashv \widehat{\alpha}_1, \widehat{\alpha}, \widehat{\beta} = \widehat{\alpha}_1$
  - after scope constraint: $\widehat{\alpha}_1, \widehat{\alpha} = \mathit{Int} \to \widehat{\alpha}_1, \widehat{\beta} = \widehat{\alpha}_1$
- $\widehat{\alpha}, \widehat{\beta}, x \vdash \widehat{\alpha} \simeq x \to \widehat{\beta}$

Strategy

Type Sanitization

Given $\widehat{\alpha}, \tau$, solve unification variables in τ out of scope of $\widehat{\alpha}$ by fresh unification variables that in that scope of $\widehat{\alpha}$.

Example

- $\widehat{\alpha}, \widehat{\beta} \vdash \widehat{\alpha} \simeq \mathit{Int} \to \widehat{\beta}$
  - type sanitization: $\widehat{\alpha}, \widehat{\beta} \vdash \mathit{Int} \to \widehat{\beta} \longmapsto \mathit{Int} \to \widehat{\alpha}_1 \dashv \widehat{\alpha}_1, \widehat{\alpha}, \widehat{\beta} = \widehat{\alpha}_1$
  - after scope constraint: $\widehat{\alpha}_1, \widehat{\alpha} = \mathit{Int} \to \widehat{\alpha}_1, \widehat{\beta} = \widehat{\alpha}_1$
- $\widehat{\alpha}, \widehat{\beta}, x \vdash \widehat{\alpha} \simeq x \to \widehat{\beta}$
  - type sanitization: $\widehat{\alpha}, \widehat{\beta}, x \vdash x \to \widehat{\beta} \longmapsto x \to \widehat{\alpha}_1 \dashv \widehat{\alpha}_1, \widehat{\alpha}, \widehat{\beta} = \widehat{\alpha}_1$

Type Sanitization

Given $\widehat{\alpha}, \tau$, solve unification variables in τ out of scope of $\widehat{\alpha}$ by fresh unification variables that in that scope of $\widehat{\alpha}$.

Example

- $\widehat{\alpha}, \widehat{\beta} \vdash \widehat{\alpha} \simeq \mathit{Int} \to \widehat{\beta}$
  - type sanitization: $\widehat{\alpha}, \widehat{\beta} \vdash \mathit{Int} \to \widehat{\beta} \longmapsto \mathit{Int} \to \widehat{\alpha}_1 \dashv \widehat{\alpha}_1, \widehat{\alpha}, \widehat{\beta} = \widehat{\alpha}_1$
  - after scope constraint: $\widehat{\alpha}_1, \widehat{\alpha} = \mathit{Int} \to \widehat{\alpha}_1, \widehat{\beta} = \widehat{\alpha}_1$
- $\widehat{\alpha}, \widehat{\beta}, x \vdash \widehat{\alpha} \simeq x \to \widehat{\beta}$
  - type sanitization: $\widehat{\alpha}, \widehat{\beta}, x \vdash x \to \widehat{\beta} \longmapsto x \to \widehat{\alpha}_1 \dashv \widehat{\alpha}_1, \widehat{\alpha}, \widehat{\beta} = \widehat{\alpha}_1$
  - after scope constraint: fail.

Unification

Key ideas:

- ordered typing context. scope constraint.
- judgment: $\Gamma \vdash \tau_1 \simeq \tau_2 \dashv \Theta$
- invariant: inputs are already fully substituted under current context.

Unification

Key ideas:

- ordered typing context. scope constraint.
- judgment: $\Gamma \vdash \tau_1 \simeq \tau_2 \dashv \Theta$
- invariant: inputs are already fully substituted under current context.
- strategy: type sanitization

Unification

Key ideas:

- ordered typing context. scope constraint.
- judgment: $\Gamma \vdash \tau_1 \simeq \tau_2 \dashv \Theta$
- invariant: inputs are already fully substituted under current context.
- strategy: type sanitization

...Find more explanations in the paper.

Outline

Language

Syntax

Type	σ ::=	$\widehat{\alpha} \mid e$
Expr	e ::=	$x \mid \star \mid e_1\ e_2 \mid \lambda x : \sigma.\ e \mid \Pi x : \sigma_1.\ \sigma_2$
		$\mid\ \forall x : \star.\sigma$
Monotype	τ ::=	$\{\sigma' \in \sigma, \forall \notin \sigma'\}$

- A restricted version of polymorphic types.
- We write $\forall a.a \rightarrow a$ for $\forall a : \star.a \rightarrow a$.

Language

Syntax

Type	$\sigma ::= \widehat{\alpha} \mid e$
Expr	$e ::= x \mid \star \mid e_1\ e_2 \mid \lambda x : \sigma.\ e \mid \Pi x : \sigma_1.\ \sigma_2$
	$\mid\ \forall x : \star . \sigma$
Monotype	$\tau ::= \{\sigma' \in \sigma, \forall \notin \sigma'\}$

- A restricted version of polymorphic types.
- We write $\forall a.a \to a$ for $\forall a : \star . a \to a$.
- Predictivity: universal quantifiers can only be instantiated by monotypes.

Language

Syntax

$$
\begin{array}{lrcl}
\text{Type} & \sigma & ::= & \widehat{\alpha} \mid e \\
\text{Expr} & e & ::= & x \mid \star \mid e_1\ e_2 \mid \lambda x : \sigma.\ e \mid \Pi x : \sigma_1.\ \sigma_2 \\
 & & \mid & \forall x : \star.\sigma \\
\text{Monotype} & \tau & ::= & \{\sigma' \in \sigma, \forall \notin \sigma'\}
\end{array}
$$

- A restricted version of polymorphic types.
- We write $\forall a.a \to a$ for $\forall a : \star.a \to a$.
- Predictivity: universal quantifiers can only be instantiated by monotypes.
- Unification is between monotypes.
- Unification variables can only have monotypes.

Subtyping

Polymorphic Subtyping

σ_1 is a subtype of σ_2, denoted by $\Gamma \vdash \sigma_1 \sqsubseteq \sigma_2$, if σ_1 is more polymorphic than σ_2 under Γ.

- examples:
  - $\Gamma \vdash \forall a.a \to a \sqsubseteq Int \to Int$
  - $\Gamma \vdash Int \to (\forall a.a \to a) \sqsubseteq Int \to (Int \to Int)$
  - $\Gamma \vdash (Int \to Int) \to Int \sqsubseteq (\forall a.a \to a) \to Int$

Problem

What happen if we have a unification variable on one side?

Problem

What happen if we have a unification variable on one side?

- do unification?

$$\frac{\Gamma \vdash \widehat{\alpha} \simeq \sigma \dashv \Theta}{\Gamma \vdash \widehat{\alpha} \sqsubseteq \sigma \dashv \Theta}$$

Problem

What happen if we have a unification variable on one side?

- do unification?

$$\frac{\Gamma \vdash \widehat{\alpha} \simeq \sigma \dashv \Theta}{\Gamma \vdash \widehat{\alpha} \sqsubseteq \sigma \dashv \Theta}$$

- unification variables can only be solved by monotypes!

Problem

What happen if we have a unification variable on one side?

- do unification?

$$\frac{\Gamma \vdash \widehat{\alpha} \simeq \sigma \dashv \Theta}{\Gamma \vdash \widehat{\alpha} \sqsubseteq \sigma \dashv \Theta}$$

- unification variables can only be solved by monotypes!
- however, we cannot restrict σ to be a monotype

$$\Gamma \vdash \widehat{\alpha} \sqsubseteq \Pi x : (\forall y.y \to y).\ \mathit{Int}$$

with solution $\widehat{\alpha} = \Pi x : (\mathit{Int} \to \mathit{Int}).\ \mathit{Int}$

Problem

What happen if we have a unification variable on one side?

- do unification?

$$\frac{\Gamma \vdash \widehat{\alpha} \simeq \sigma \dashv \Theta}{\Gamma \vdash \widehat{\alpha} \sqsubseteq \sigma \dashv \Theta}$$

- unification variables can only be solved by monotypes!
- however, we cannot restrict σ to be a monotype

$$\Gamma \vdash \widehat{\alpha} \sqsubseteq \Pi x : (\forall y.y \to y).\ Int$$

with solution $\widehat{\alpha} = \Pi x : (Int \to Int).\ Int$

- again, we cannot destruct pi type because of type dependency.

$$\Gamma \vdash \widehat{\alpha} \sqsubseteq \Pi x : \star.\ x$$

Problem

- $\Gamma \vdash \widehat{\alpha} \sqsubseteq \Pi x : (\forall y.y \to y).\ \mathit{Int}$
  with solution $\widehat{\alpha} = \Pi x : (\mathit{Int} \to \mathit{Int}).\ \mathit{Int}$

Problem

- $\Gamma \vdash \widehat{\alpha} \sqsubseteq \Pi x : (\forall y.y \to y).\ Int$
  with solution $\widehat{\alpha} = \Pi x : (Int \to Int).\ Int$
- Observation: when the unification variable is on the left, even though there can be polymorphic components on the right, those polymorphic components must appear contra-variantly.

Problem

- $\Gamma \vdash \widehat{\alpha} \sqsubseteq \Pi x : (\forall y.y \to y).\ Int$
  with solution $\widehat{\alpha} = \Pi x : (Int \to Int).\ Int$
- Observation: when the unification variable is on the left, even though there can be polymorphic components on the right, those polymorphic components must appear contra-variantly.
- similar observation for when the unification variable on the right, polymorphic components must appear co-variantly.

Problem

- $\Gamma \vdash \widehat{\alpha} \sqsubseteq \Pi x : (\forall y.y \to y).\ Int$
  with solution $\widehat{\alpha} = \Pi x : (Int \to Int).\ Int$
- Observation: when the unification variable is on the left, even though there can be polymorphic components on the right, those polymorphic components must appear contra-variantly.
- similar observation for when the unification variable on the right, polymorphic components must appear co-variantly.
- $\widehat{\alpha} = \Pi x : (Bool \to Bool).\ Int$

Problem

- $\Gamma \vdash \widehat{\alpha} \sqsubseteq \Pi x : (\forall y. y \to y).\ Int$
  with solution $\widehat{\alpha} = \Pi x : (Int \to Int).\ Int$
- Observation: when the unification variable is on the left, even though there can be polymorphic components on the right, those polymorphic components must appear contra-variantly.
- similar observation for when the unification variable on the right, polymorphic components must appear co-variantly.
- $\widehat{\alpha} = \Pi x : (Bool \to Bool).\ Int$
- $\widehat{\alpha} = \Pi x : (\widehat{\alpha}_1 \to \widehat{\alpha}_1).\ Int$

Problem

- $\Gamma \vdash \widehat{\alpha} \sqsubseteq \Pi x : (\forall y.y \rightarrow y).\ Int$
  with solution $\widehat{\alpha} = \Pi x : (Int \rightarrow Int).\ Int$
- Observation: when the unification variable is on the left, even though there can be polymorphic components on the right, those polymorphic components must appear contra-variantly.
- similar observation for when the unification variable on the right, polymorphic components must appear co-variantly.
- $\widehat{\alpha} = \Pi x : (Bool \rightarrow Bool).\ Int$
- $\widehat{\alpha} = \Pi x : (\widehat{\alpha}_1 \rightarrow \widehat{\alpha}_1).\ Int$
- How to turn $\Pi x : (\forall y.y \rightarrow y).\ Int$ into $\Pi x : (\widehat{\alpha}_1 \rightarrow \widehat{\alpha}_1).\ Int$

Strategy

- How to turn $\Pi x : (\forall y . y \to y).\ Int$ into $\Pi x : (\widehat{\alpha}_1 \to \widehat{\alpha}_1).\ Int$
- we can always replace universal quantifiers that appear contra-variantly by a fresh unification variable.

Strategy

- How to turn $\Pi x : (\forall y.y \to y).\ Int$ into $\Pi x : (\widehat{\alpha}_1 \to \widehat{\alpha}_1).\ Int$
- we can always replace universal quantifiers that appear contra-variantly by a fresh unification variable.
- our solution: for subtyping problem between $\widehat{\alpha}$ and σ, we sanitize the contra-variant universal quantifiers in σ before we use unification.

Strategy

Polymorphic Type Sanitization

Given $\widehat{\alpha}, \sigma$, remove universal quantifiers appearing contra-variantly, and replace corresponding type variables by a fresh unification variable.

Strategy

Polymorphic Type Sanitization

Given $\widehat{\alpha}, \sigma$, remove universal quantifiers appearing contra-variantly, and replace corresponding type variables by a fresh unification variable.

Example

- $\widehat{\alpha} \vdash \widehat{\alpha} \sqsubseteq (\forall a.a \rightarrow a) \rightarrow Int$

Strategy

Polymorphic Type Sanitization

Given $\widehat{\alpha}, \sigma$, remove universal quantifiers appearing contra-variantly, and replace corresponding type variables by a fresh unification variable.

Example

- $\widehat{\alpha} \vdash \widehat{\alpha} \sqsubseteq (\forall a.a \rightarrow a) \rightarrow Int$
  - polymorphic type sanitization:
    $\widehat{\alpha} \vdash (\forall a.a \rightarrow a) \rightarrow Int \longmapsto (\widehat{\alpha}_1 \rightarrow \widehat{\alpha}_1) \rightarrow Int \dashv \widehat{\alpha}_1, \widehat{\alpha}$

Strategy

Polymorphic Type Sanitization

Given $\widehat{\alpha}, \sigma$, remove universal quantifiers appearing contra-variantly, and replace corresponding type variables by a fresh unification variable.

Example

- $\widehat{\alpha} \vdash \widehat{\alpha} \sqsubseteq (\forall a.a \to a) \to \mathit{Int}$
  - polymorphic type sanitization:
    $\widehat{\alpha} \vdash (\forall a.a \to a) \to \mathit{Int} \longmapsto (\widehat{\alpha}_1 \to \widehat{\alpha}_1) \to \mathit{Int} \dashv \widehat{\alpha}_1, \widehat{\alpha}$
  - after unification: $\widehat{\alpha}_1, \widehat{\alpha} = (\widehat{\alpha}_1 \to \widehat{\alpha}_1) \to \mathit{Int}$

Strategy

Polymorphic Type Sanitization

Given $\widehat{\alpha}, \sigma$, remove universal quantifiers appearing contra-variantly, and replace corresponding type variables by a fresh unification variable.

Example

- $\widehat{\alpha} \vdash \widehat{\alpha} \sqsubseteq (\forall a.a \to a) \to Int$
  - polymorphic type sanitization:
    $\widehat{\alpha} \vdash (\forall a.a \to a) \to Int \longmapsto (\widehat{\alpha}_1 \to \widehat{\alpha}_1) \to Int \dashv \widehat{\alpha}_1, \widehat{\alpha}$
  - after unification: $\widehat{\alpha}_1, \widehat{\alpha} = (\widehat{\alpha}_1 \to \widehat{\alpha}_1) \to Int$
- $\widehat{\alpha} \vdash \widehat{\alpha} \sqsubseteq (\forall a.a \to a)$

Strategy

Polymorphic Type Sanitization

Given $\widehat{\alpha}, \sigma$, remove universal quantifiers appearing contra-variantly, and replace corresponding type variables by a fresh unification variable.

Example

- $\widehat{\alpha} \vdash \widehat{\alpha} \sqsubseteq (\forall a.a \to a) \to Int$
  - polymorphic type sanitization:
    $\widehat{\alpha} \vdash (\forall a.a \to a) \to Int \longmapsto (\widehat{\alpha}_1 \to \widehat{\alpha}_1) \to Int \dashv \widehat{\alpha}_1, \widehat{\alpha}$
  - after unification: $\widehat{\alpha}_1, \widehat{\alpha} = (\widehat{\alpha}_1 \to \widehat{\alpha}_1) \to Int$
- $\widehat{\alpha} \vdash \widehat{\alpha} \sqsubseteq (\forall a.a \to a)$
  - polymorphic type sanitization fail.

Strategy

Polymorphic Type Sanitization

Given $\widehat{\alpha}, \sigma$, remove universal quantifiers appearing contra-variantly, and replace corresponding type variables by a fresh unification variable.

Example

- $\widehat{\alpha} \vdash \widehat{\alpha} \sqsubseteq (\forall a.a \to a) \to Int$
  - polymorphic type sanitization:
    $\widehat{\alpha} \vdash (\forall a.a \to a) \to Int \longmapsto (\widehat{\alpha}_1 \to \widehat{\alpha}_1) \to Int \dashv \widehat{\alpha}_1, \widehat{\alpha}$
  - after unification: $\widehat{\alpha}_1, \widehat{\alpha} = (\widehat{\alpha}_1 \to \widehat{\alpha}_1) \to Int$
- $\widehat{\alpha} \vdash \widehat{\alpha} \sqsubseteq (\forall a.a \to a)$
  - polymorphic type sanitization fail.
- $\widehat{\alpha}, \widehat{\beta} \vdash \widehat{\alpha} \sqsubseteq Int \to \widehat{\beta}$

Strategy

Polymorphic Type Sanitization

Given $\widehat{\alpha}, \sigma$, remove universal quantifiers appearing contra-variantly, and replace corresponding type variables by a fresh unification variable.

Example

- $\widehat{\alpha} \vdash \widehat{\alpha} \sqsubseteq (\forall a.a \to a) \to Int$
  - polymorphic type sanitization:
    $\widehat{\alpha} \vdash (\forall a.a \to a) \to Int \longmapsto (\widehat{\alpha}_1 \to \widehat{\alpha}_1) \to Int \dashv \widehat{\alpha}_1, \widehat{\alpha}$
  - after unification: $\widehat{\alpha}_1, \widehat{\alpha} = (\widehat{\alpha}_1 \to \widehat{\alpha}_1) \to Int$
- $\widehat{\alpha} \vdash \widehat{\alpha} \sqsubseteq (\forall a.a \to a)$
  - polymorphic type sanitization fail.
- $\widehat{\alpha}, \widehat{\beta} \vdash \widehat{\alpha} \sqsubseteq Int \to \widehat{\beta}$
  - polymorphic type sanitization: $\widehat{\alpha}, \widehat{\beta} \vdash Int \to \widehat{\beta} \longmapsto Int \to \widehat{\beta} \dashv \widehat{\alpha}, \widehat{\beta}$

Strategy

Polymorphic Type Sanitization

Given $\widehat{\alpha}, \sigma$, remove universal quantifiers appearing contra-variantly, and replace corresponding type variables by a fresh unification variable.

Example

- $\widehat{\alpha} \vdash \widehat{\alpha} \sqsubseteq (\forall a.a \to a) \to Int$
  - polymorphic type sanitization:
    $\widehat{\alpha} \vdash (\forall a.a \to a) \to Int \longmapsto (\widehat{\alpha}_1 \to \widehat{\alpha}_1) \to Int \dashv \widehat{\alpha}_1, \widehat{\alpha}$
  - after unification: $\widehat{\alpha}_1, \widehat{\alpha} = (\widehat{\alpha}_1 \to \widehat{\alpha}_1) \to Int$
- $\widehat{\alpha} \vdash \widehat{\alpha} \sqsubseteq (\forall a.a \to a)$
  - polymorphic type sanitization fail.
- $\widehat{\alpha}, \widehat{\beta} \vdash \widehat{\alpha} \sqsubseteq Int \to \widehat{\beta}$
  - polymorphic type sanitization: $\widehat{\alpha}, \widehat{\beta} \vdash Int \to \widehat{\beta} \longmapsto Int \to \widehat{\beta} \dashv \widehat{\alpha}, \widehat{\beta}$
  - unification: $\widehat{\alpha}_1, \widehat{\alpha} = Int \to \widehat{\alpha}_1, \widehat{\beta} = \widehat{\alpha}_1$

Strategy

Polymorphic Type Sanitization

Given $\widehat{\alpha}, \sigma$, remove universal quantifiers appearing contra-variantly, and replace corresponding type variables by a fresh unification variable.

Example

- $\widehat{\alpha} \vdash \widehat{\alpha} \sqsubseteq (\forall a.a \to a) \to Int$
  - polymorphic type sanitization:
    $\widehat{\alpha} \vdash (\forall a.a \to a) \to Int \longmapsto (\widehat{\alpha}_1 \to \widehat{\alpha}_1) \to Int \dashv \widehat{\alpha}_1, \widehat{\alpha}$
  - after unification: $\widehat{\alpha}_1, \widehat{\alpha} = (\widehat{\alpha}_1 \to \widehat{\alpha}_1) \to Int$
- $\widehat{\alpha} \vdash \widehat{\alpha} \sqsubseteq (\forall a.a \to a)$
  - polymorphic type sanitization fail.
- $\widehat{\alpha}, \widehat{\beta} \vdash \widehat{\alpha} \sqsubseteq Int \to \widehat{\beta}$
  - polymorphic type sanitization: $\widehat{\alpha}, \widehat{\beta} \vdash Int \to \widehat{\beta} \longmapsto Int \to \widehat{\beta} \dashv \widehat{\alpha}, \widehat{\beta}$
  - unification: $\widehat{\alpha}_1, \widehat{\alpha} = Int \to \widehat{\alpha}_1, \widehat{\beta} = \widehat{\alpha}_1$
- Find more explanations in the paper!

Outline

Related Work

- Powerful but complicated unification algorithms for dependent types:
  - Ziliani, B., Sozeau, M. (2015, August) [3]; Elliott, C. (1989). [4]; Abel, A., Pientka, B. (2011, June) [5]
- Complete and easy unification/subtyping algorithm for simple types and System F types:
  - Hindley-Milner algorithm [6] [7]; Dunfield, J., Krishnaswami, N. R. (2013, September). [8]; Jones, S. P., Vytiniotis, D., Weirich, S., Shields, M. (2007) [9];
- Dependent type systems with alpha-equality based type checking:
  - type-level computation by explicit casts [10] [11] [12] [13]

[3] Ziliani, Beta, and Matthieu Sozeau. "A unification algorithm for Coq featuring universe polymorphism and overloading." ACM SIGPLAN Notices. Vol. 50. No. 9. ACM, 2015.

[4] Elliott, Conal. "Higher-order unification with dependent function types." Rewriting Techniques and Applications. Springer Berlin/Heidelberg, 1989.

[5] Abel, Andreas, and Brigitte Pientka. "Higher-order dynamic pattern unification for dependent types and records." International Conference on Typed Lambda Calculi and Applications. Springer Berlin Heidelberg, 2011.

[6] Damas, Luis, and Robin Milner. "Principal type-schemes for functional programs." Proceedings of the 9th ACM SIGPLAN-SIGACT symposium on Principles of programming languages. ACM, 1982.

[7] Hindley, Roger. "The principal type-scheme of an object in combinatory logic." Transactions of the american mathematical society 146 (1969): 29-60.

[8] Dunfield, Joshua, and Neelakantan R. Krishnaswami. "Complete and easy bidirectional typechecking for higher-rank polymorphism." ACM SIGPLAN Notices. Vol. 48. No. 9. ACM, 2013.

[9] Jones, Simon Peyton, et al. "Practical type inference for arbitrary-rank types." Journal of functional programming 17.01

Conclusion

- Strategy: a both simple to understand and simple to implement strategy called *type sanitization*
- Algorithm: A simple and complete alpha-equality based unification algorithm
- Extension: *polymorphic type sanitization* to deal with polymorphic subtyping.
- Meta-theory: proof sketches.

Thanks for listening!