Bidirectional Higher-rank Polymorphism with Intersection and Union Types

Shengyi Jiang, Chen Cui, Bruno C. d. S. Oliveira

February 13, 2025

HKU PLGroup

Table of Contents

Table of Contents

Higher-rank Polymorphism

Parametric polymorphism allows a single piece of code to be given a "generic" type (a.k.a polymorphic type), using variables in place of actual types, and then instantiated with particular types as needed

$$\lambda x.\ x : \forall a.\ a \to a$$

In Hindley-Milner type system, polymorphic types are restricted to the form $\forall \bar{a}.A$, where A has no more foralls. This restriction prevents it from expressing functions that take a polymorphic function as an argument:

$$f : (\forall a.\ a \to a) \to \mathtt{Int} \to \mathtt{Int}$$

Higher-rank Polymorphism

Parametric polymorphism allows a single piece of code to be given a "generic" type (a.k.a polymorphic type), using variables in place of actual types, and then instantiated with particular types as needed

$$\lambda x.\ x : \forall a.\ a \to a$$

Higher-rank polymorphism: $\forall$ quantifiers can appear inside the function types

Rank-n polymorphism: function type with a Rank-n-1 type as an argument is allowed

- Rank-1 type: $\forall a.\ a \to a \to \texttt{Int}$
- Rank-2 type: $(\forall a.\ a \to a) \to \texttt{Int}$

HKU PLGroup

Intersection and Union Type

$e : A \sqcap B$ means e has both type A and B

$e : A \sqcup B$ means e has type A or B

$$\frac{\Psi \vdash A \leqslant B_1 \quad \Psi \vdash A \leqslant B_2}{\Psi \vdash A \leqslant B_1 \sqcap B_2} \leqslant\sqcap\mathrm{R} \qquad \frac{\Psi \vdash A_1 \leqslant B}{\Psi \vdash A_1 \sqcap A_2 \leqslant B} \leqslant\sqcap\mathrm{L}_1 \qquad \frac{\Psi \vdash A_2 \leqslant B}{\Psi \vdash A_1 \sqcap A_2 \leqslant B} \leqslant\sqcap\mathrm{L}_2$$

$$\frac{\Psi \vdash A_1 \leqslant B \quad \Psi \vdash A_2 \leqslant B}{\Psi \vdash A_1 \sqcup A_2 \leqslant B} \leqslant\sqcup\mathrm{L} \qquad \frac{\Psi \vdash A \leqslant B_1}{\Psi \vdash A \leqslant B_1 \sqcup B_2} \leqslant\sqcup\mathrm{R}_1 \qquad \frac{\Psi \vdash A \leqslant B_2}{\Psi \vdash A \leqslant B_1 \sqcup B_2} \leqslant\sqcup\mathrm{R}_2$$

Feature Interaction

Higher-Rank Polymorphism, Intersection and Union Types, Explicit Type Application

- Core features of several mainstream languages, e.g., Scala and TypeScript;
- Expressive enough to type a large portion of dynamic language patterns (Castagna et al. 2024)
- Parametric polymorphism and intersection types are both important mechanisms of polymorphism;
- Explicit-type applications allow programmers to provide complex and unambiguous instantiations;

Feature Interaction

Higher-Rank Polymorphism, Intersection and Union Types, Explicit Type Application

- Core features of several mainstream languages, e.g., Scala and TypeScript;
  - Heterogeneous list, mix-in patterns, overloading
  - DOT calculus
- Expressive enough to type a large portion of dynamic language patterns (Castagna et al. 2024)
- Parametric polymorphism and intersection types are both important mechanisms of polymorphism;
- Explicit-type applications allow programmers to provide complex and unambiguous instantiations;

HKU PLGroup

Feature Interaction

Higher-Rank Polymorphism, Intersection and Union Types, Explicit Type Application

- Core features of several mainstream languages, e.g., Scala and TypeScript;
- Expressive enough to type a large portion of dynamic language patterns (Castagna et al. 2024)
- Parametric polymorphism and intersection types are both important mechanisms of polymorphism;
- Explicit-type applications allow programmers to provide complex and unambiguous instantiations;
  - subtyping of implicit System F is undecidable
  - resolve ambiguity (`show (read @Int "5")`)
  - already supported by most languages (`f<Int>(5)`)

HKU PLGroup

Type Inference

By type inference, we mean

- implicit instantiation, which in general consists of two cases
  - $\mathsf{id} : \forall a.a \to a \vdash \mathsf{id}\ 1$
  - $\mathsf{f} : ((\forall a.a \to a) \to \mathtt{Int}) \to \mathtt{Int}, \mathsf{g} : (\mathtt{Int} \to \mathtt{Int}) \to \mathtt{Int} \vdash \mathsf{f}\ \mathsf{g}$

Type Inference

By type inference, we mean

- implicit instantiation, which in general consists of two cases
  - $\text{id} : \forall a.a \to a \vdash \text{id}\ \{\texttt{Int}\}\ 1$
  - $\text{f} : ((\forall a.a \to a) \to \texttt{Int}) \to \texttt{Int},\ \text{g} : (\texttt{Int} \to \texttt{Int}) \to \texttt{Int} \vdash \text{f g}$ $(\texttt{Int} \to \texttt{Int}) \to \texttt{Int} \leqslant (\forall a.a \to a) \to \texttt{Int}$ (polymorphic subtyping)

HKU PLGroup

Type Inference

By type inference, we mean

- implicit instantiation, which in general consists of two cases
  - $\mathsf{id} : \forall a.a \to a \vdash \mathsf{id}\ \{\texttt{Int}\}\ 1$
  - $\mathsf{f} : ((\forall a.a \to a) \to \texttt{Int}) \to \texttt{Int},\ \mathsf{g} : (\texttt{Int} \to \texttt{Int}) \to \texttt{Int} \vdash \mathsf{f}\ \mathsf{g}$
    $(\texttt{Int} \to \texttt{Int}) \to \texttt{Int} \leqslant (\forall a.a \to a) \to \texttt{Int}$ (polymorphic subtyping)

$$\frac{\Psi \vdash [\tau/a]A \leqslant B \quad \Psi \vdash \tau}{\Psi \vdash \forall a.A \leqslant B}$$

HKU PLGroup

Type Inference

By type inference, we mean

- implicit instantiation, which in general consists of two cases
  - $\mathsf{id} : \forall a.a \to a \vdash \mathsf{id}\ \{\mathtt{Int}\}\ 1$
  - $\mathsf{f} : ((\forall a.a \to a) \to \mathtt{Int}) \to \mathtt{Int},\ \mathsf{g} : (\mathtt{Int} \to \mathtt{Int}) \to \mathtt{Int} \vdash \mathsf{f}\ \mathsf{g}$
    $(\mathtt{Int} \to \mathtt{Int}) \to \mathtt{Int} \leqslant (\forall a.a \to a) \to \mathtt{Int}$ (polymorphic subtyping)

  There are two kinds of instantiation:
  - Predicative: type for instantiation can only be monomorphic types
  - Impredicative: type for instantiation can be polymorphic types, e.g.
    $\mathsf{id} : \forall a.a \to a \vdash \mathsf{id}\ \{\forall a.a \to a\}\ \mathsf{id}$

Type Inference

By type inference, we mean

- implicit instantiation, which in general consists of two cases
  - $\mathsf{id} : \forall a.a \to a \vdash \mathsf{id}\ \{\mathtt{Int}\}\ 1$
  - $\mathsf{f} : ((\forall a.a \to a) \to \mathtt{Int}) \to \mathtt{Int},\ \mathsf{g} : (\mathtt{Int} \to \mathtt{Int}) \to \mathtt{Int} \vdash \mathsf{f}\ \mathsf{g}$
    $(\mathtt{Int} \to \mathtt{Int}) \to \mathtt{Int} \leqslant (\forall a.a \to a) \to \mathtt{Int}$ (polymorphic subtyping)
- unannotated monotype function
  $(\lambda x.x : \mathtt{Int} \to \mathtt{Int})\ 1$

HKU PLGroup

Type Inference

By type inference, we mean

- implicit instantiation, which in general consists of two cases
  - $\mathrm{id} : \forall a.a \to a \vdash \mathrm{id}\ \{\mathtt{Int}\}\ 1$
  - $\mathrm{f} : ((\forall a.a \to a) \to \mathtt{Int}) \to \mathtt{Int},\ \mathrm{g} : (\mathtt{Int} \to \mathtt{Int}) \to \mathtt{Int} \vdash \mathrm{f}\ \mathrm{g}$ $(\mathtt{Int} \to \mathtt{Int}) \to \mathtt{Int} \leqslant (\forall a.a \to a) \to \mathtt{Int}$ (polymorphic subtyping)
- unannotated monotype function $(\lambda x.x : \mathtt{Int} \to \mathtt{Int})\ 1$

A bidirectional framework

- $e \Rightarrow A$ inference mode
- $e \Leftarrow A$ checking mode

Table of Contents

Overview

Philosophy

- infer easy (predicative) instantiations
- use explicit annotations for hard (impredicative) instantiations

$F^e_{\sqcup\sqcap}$, an extension of F^e (Zhao et al. 2022) with first-class intersection and union type

More Disciplined Treatment

Overview, compared with TypeScript

Based on its behavior, the type inference of TypeScript seems to be loosely based on DK's (Dunfield et al. 2013) algorithm, with certain ad-hoc design choices

$F^e_{\sqcup\sqcap}$ provides more disciplined treatment and has the following advantages

- Order-relevant quantifier and checking subsumption
- More complete overloading
- Less syntactic restriction
- Intersection introduction rule
- Monotype function inference
- Guaranteed monotype-instantiation inference

More Disciplined Treatment

Order-relevant quantifier and checking subsumption

Order-irrelevant quantifer is known to be incompatible with explicit type application

$\forall a.\, \forall b.\, a \to b \to a \leqslant \forall a.\, \mathtt{Int} \to a \to \mathtt{Int}$

However, after instantiating the first quantifier to Bool

$\forall b.\, \mathtt{Bool} \to b \to \mathtt{Bool} \nleqslant \mathtt{Int} \to \mathtt{Bool} \to \mathtt{Int}$

```
var f: (k: <A>(_:number)=>(_:A)=>number) => (_:boolean) => number = k => k(3)
var h: (k: <A,B>(_:A)=>(_:B)=>A) => (b: boolean) => number = k => f(k)
var g: (k: <A>(_:number)=>(_:A)=>number) => (_:boolean) => number = k => k<boolean>(3)

var ex1: (k: <A,B>(_:A)=>(_:B)=>A) => (_:boolean) => number = k => g(k)
var ex2: (k: <A,B>(_:A)=>(_:B)=>A) => (_:boolean) => number = k => k<boolean>(3) // rejected!
```

- `f(k)` is rejected in the first place by $F^{e}_{\sqcup\sqcap}$

Design of F^e

Type variables	a, b	Subtype variables	$\tilde{a}, \tilde{b}$
Declarative types	A, B, C	$::=$	$1 \mid a \mid \tilde{a} \mid \forall a.\ A \mid A \to B \mid \top \mid \bot$
Monotypes	τ	$::=$	$1 \mid a \mid \tau_1 \to \tau_2$
Expressions	e	$::=$	$(\!\mid\ \mid\!) \mid x \mid \lambda x.e \mid e_1\ e_2 \mid \Lambda a.e : A \mid e\ @A$

- Order-relevant quantifiers

$$\forall a.\forall b.a \to b \not\leqslant \forall b.\forall a.a \to b$$

- No unused quantifier

$\forall a.\texttt{Int}$ is not a well-formed type

HKU PLGroup

Design of F^e

Type variables	a, b	Subtype variables	$\tilde{a}, \tilde{b}$
Declarative types	A, B, C	$::=$	$1 \mid a \mid \tilde{a} \mid \forall a.\ A \mid A \to B \mid \top \mid \bot$
Monotypes	τ	$::=$	$1 \mid a \mid \tau_1 \to \tau_2$
Expressions	e	$::=$	$(\!\mid\ \mid\!) \mid x \mid \lambda x.e \mid e_1\ e_2 \mid \Lambda a.e : A \mid e\ @A$

Modification in subtyping

$$\frac{\Psi \vdash [\tau/a]A \leqslant B \quad \Psi \vdash \tau}{\Psi \vdash \forall a.A \leqslant B} \qquad \frac{\Psi, b \vdash A \leqslant B}{\Psi \vdash A \leqslant \forall b.B}$$

Design of F^e

Type variables	a, b	Subtype variables	$\tilde{a}, \tilde{b}$
Declarative types	A, B, C	$::=$	$1 \mid a \mid \tilde{a} \mid \forall a.\ A \mid A \to B \mid \top \mid \bot$
Monotypes	τ	$::=$	$1 \mid a \mid \tau_1 \to \tau_2$
Expressions	e	$::=$	$(\!\!\mid\ \mid\!\!) \mid x \mid \lambda x.e \mid e_1\ e_2 \mid \Lambda a.e : A \mid e\ @A$

Modification in subtyping

$$\frac{\Psi \vdash [\tau/a]A \leqslant B \quad \Psi \vdash \tau \quad B \neq \forall.*}{\Psi \vdash \forall a.A \leqslant B} \qquad \frac{\Psi, \tilde{a} \vdash [\tilde{a}/a]A \leqslant [\tilde{a}/b]B}{\Psi \vdash \forall a.A \leqslant \forall b.B}$$

HKU PLGroup

Design of F^e

Type variables	a, b	Subtype variables	$\tilde{a}, \tilde{b}$	
Declarative types	A, B, C	$::=$	$1 \mid a \mid \tilde{a} \mid \forall a.\ A \mid A \to B \mid \top \mid \bot$	
Monotypes	τ	$::=$	$1 \mid a \mid \tau_1 \to \tau_2$	
Expressions	e	$::=$	$(\!|\ |\!) \mid x \mid \lambda x.e \mid e_1\ e_2 \mid \Lambda a.e : A \mid e\ @A$	

Modification in subtyping

$$\frac{\Psi \vdash [\tau/a]A \leqslant B \quad \Psi \vdash \tau \quad B \neq \forall.*}{\Psi \vdash \forall a.A \leqslant B} \qquad \frac{\Psi, \tilde{a} \vdash [\tilde{a}/a]A \leqslant [\tilde{a}/b]B}{\Psi \vdash \forall a.A \leqslant \forall b.B}$$

Modification in well-formedness

$$\frac{\Psi, a \vdash A \quad a \in \mathrm{fv}(A)}{\Psi \vdash \forall a.A} \qquad \frac{\Psi, a \vdash e : A \quad a \in \mathrm{fv}(A)}{\Psi \vdash \Lambda a.e : A}$$

HKU PLGroup

Syntax

Type variables	a, b	Subtype variables	$\tilde{a}, \tilde{b}$
Declarative types	A, B, C	$::=$	$\mathbb{1} \mid a \mid \tilde{a} \mid \forall a.\ A \mid A \to B \mid \top \mid \bot \mid A \sqcap B \mid A \sqcup B$
Monotypes	τ	$::=$	$\mathbb{1} \mid a \mid \tau_1 \to \tau_2$
Expressions	e	$::=$	$(\!\mid\,\mid\!) \mid x \mid \lambda x.e \mid e_1\ e_2 \mid e : A \mid \Lambda a.e : A \mid e\ @A$

$$\frac{\Psi, a \vdash A \quad a \in^s \mathrm{fv}(A)}{\Psi \vdash \forall a.A} \qquad \frac{\Psi, a \vdash e : A \quad a \in^s \mathrm{fv}(A)}{\Psi \vdash \Lambda a.e : A}$$

$$\frac{}{a \in^s a} \qquad \frac{a \in^s \mathrm{fv}(A)}{a \in^s \mathrm{fv}(A \to B)} \qquad \frac{a \in^s \mathrm{fv}(B)}{a \in^s \mathrm{fv}(A \to B)} \qquad \frac{a \in^s \mathrm{fv}(B) \quad a \neq b}{a \in^s \mathrm{fv}(\forall b.B)}$$

$$\frac{a \in^s \mathrm{fv}(A_1) \quad a \in^s \mathrm{fv}(A_2)}{a \in^s \mathrm{fv}(A_1 \sqcap A_2)} \qquad \frac{a \in^s \mathrm{fv}(A_1)}{a \in^s \mathrm{fv}(A_1 \sqcup A_2)} \qquad \frac{a \in^s \mathrm{fv}(A_2)}{a \in^s \mathrm{fv}(A_1 \sqcup A_2)}$$

HKU PLGroup

Syntax

Type variables	a, b	Subtype variables	$\tilde{a}, \tilde{b}$
Declarative types	A, B, C	$::=$	$\mathbb{1} \mid a \mid \tilde{a} \mid \forall a.\ A \mid A \to B \mid \top \mid \bot \mid A \sqcap B \mid A \sqcup B$
Monotypes	τ	$::=$	$\mathbb{1} \mid a \mid \tau_1 \to \tau_2$
Expressions	e	$::=$	$(\mid) \mid x \mid \lambda x.e \mid e_1\ e_2 \mid e : A \mid \Lambda a.e : A \mid e\ @A$

$$\frac{\Psi, a \vdash A \quad a \in^{s} \mathrm{fv}(A)}{\Psi \vdash \forall a.A} \qquad \frac{\Psi, a \vdash e : A \quad a \in^{s} \mathrm{fv}(A)}{\Psi \vdash \Lambda a.e : A}$$

$\forall a.((a \to a) \sqcup (\mathtt{Int} \to \mathtt{Int}))$ is regarded as a well-formed type

$\forall a.((a \to a) \sqcap (\mathtt{Int} \to \mathtt{Int}))$ is NOT regarded as a well-formed type

HKU PLGroup

Subtyping Rules

$$\frac{}{\Psi \vdash \mathbb{1} \leqslant \mathbb{1}} \leqslant\texttt{Unit} \qquad \frac{\vdash \Psi \quad a \in \Psi}{\Psi \vdash a \leqslant a} \leqslant\texttt{Var} \qquad \frac{\vdash \Psi \quad \tilde{a} \in \Psi}{\Psi \vdash \tilde{a} \leqslant \tilde{a}} \leqslant\texttt{Svar}$$

$$\frac{\Psi \vdash A}{\Psi \vdash \bot \leqslant A} \leqslant\bot \qquad \frac{\Psi \vdash A}{\Psi \vdash A \leqslant \top} \leqslant\top \qquad \frac{\Psi \vdash B_1 \leqslant A_1 \quad \Psi \vdash A_2 \leqslant B_2}{\Psi \vdash A_1 \to A_2 \leqslant B_1 \to B_2} \leqslant\to$$

$$\frac{\Psi \vdash \tau \quad \Psi \vdash [\tau/a]A \leqslant B \quad a \in^s \mathsf{fv}(A) \quad B?}{\Psi \vdash \forall a.A \leqslant B} \leqslant\forall\mathtt{L}$$

$$\frac{\Psi, \tilde{a} \vdash [\tilde{a}/a]A \leqslant [\tilde{a}/b]B \quad a \in^s \mathsf{fv}(A) \quad b \in^s \mathsf{fv}(B)}{\Psi \vdash \forall a.A \leqslant \forall b.B} \leqslant\forall$$

$$\frac{\Psi \vdash A \leqslant B_1 \quad \Psi \vdash A \leqslant B_2}{\Psi \vdash A \leqslant B_1 \sqcap B_2} \leqslant\sqcap\mathtt{R} \qquad \frac{\Psi \vdash A_1 \leqslant B}{\Psi \vdash A_1 \sqcap A_2 \leqslant B} \leqslant\sqcap\mathtt{L}_1 \qquad \frac{\Psi \vdash A_2 \leqslant B}{\Psi \vdash A_1 \sqcap A_2 \leqslant B} \leqslant\sqcap\mathtt{L}_2$$

$$\frac{\Psi \vdash A_1 \leqslant B \quad \Psi \vdash A_2 \leqslant B}{\Psi \vdash A_1 \sqcup A_2 \leqslant B} \leqslant\sqcup\mathtt{L} \qquad \frac{\Psi \vdash A \leqslant B_1}{\Psi \vdash A \leqslant B_1 \sqcup B_2} \leqslant\sqcup\mathtt{R}_1 \qquad \frac{\Psi \vdash A \leqslant B_2}{\Psi \vdash A \leqslant B_1 \sqcup B_2} \leqslant\sqcup\mathtt{R}_2$$

HKU PLGroup

Restriction in ∀L

$B \neq \forall.*$ no longer works

✓ $\Psi \vdash A \leqslant B$: $$\dfrac{\forall a.a \to \mathtt{Int} \leqslant (\forall a.a \to \mathtt{Int}) \sqcup (\forall a.a \to \mathtt{Int})}{\forall b.\forall a.a \to b \leqslant (\forall a.a \to \mathtt{Int}) \sqcup (\forall a.a \to \mathtt{Int})} \text{ BY } \leqslant\forall\text{L}$$

✓ $\Psi \vdash B \leqslant C$: $$\dfrac{\forall a.a \to \mathtt{Int} \leqslant \forall a.a \to \mathtt{Int}}{(\forall a.a \to \mathtt{Int}) \sqcup (\forall a.a \to \mathtt{Int}) \leqslant \forall a.a \to \mathtt{Int}} \text{ BY } \leqslant\sqcup\text{L}$$

✗ $\Psi \vdash A \not\leqslant C$: $\forall b.\forall a.a \to b \not\leqslant \forall a.a \to \mathtt{Int}$

∀L can only be applied if we can make sure the ∀ rule should not be applied to every component of the type on the RHS.

- In the above example, $(\forall a.a \to \mathtt{Int})$ is a component of RHS and $\forall a.a \to \mathtt{Int} \leqslant \forall a.a \to \mathtt{Int}$ will use the ∀ rule.

By disallowing the invocation of ∀L rule when RHS is $* \sqcup *$ or $* \sqcap *$, we can avoid such a problem.

HKU PLGroup

Restriction in $\forall$L

A naive one

$$B \neq \forall.* \quad \wedge \quad B \neq * \sqcap * \quad \wedge \quad B \neq * \sqcup *$$

It works, though too restrictive

$$\forall a.(a \to \texttt{Int}) \sqcup (a \to \texttt{Bool}) \nleqslant (\texttt{Int} \to \texttt{Int}) \sqcup (\texttt{Int} \to \texttt{Bool})$$

Restriction in ∀L

A naive one

$$B \neq \forall.* \quad \wedge \quad B \neq * \sqcap * \quad \wedge \quad B \neq * \sqcup *$$

It works, though too restrictive

$$\forall a.(a \to \texttt{Int}) \sqcup (a \to \texttt{Bool}) \nleqslant (\texttt{Int} \to \texttt{Int}) \sqcup (\texttt{Int} \to \texttt{Bool})$$

We always decompose the RHS first, even though the ∀ rule cannot be triggered later.

$$\forall a.(a \to \texttt{Int}) \sqcup (a \to \texttt{Bool}) \nleqslant (\texttt{Int} \to \texttt{Int})$$
$$\forall a.(a \to \texttt{Int}) \sqcup (a \to \texttt{Bool}) \nleqslant (\texttt{Int} \to \texttt{Bool})$$

Restriction in $\forall$L

A better one

$$\frac{}{\mathbb{1}^{<>\forall.*}} \quad \frac{}{\bot^{<>\forall.*}} \quad \frac{}{\top^{<>\forall.*}} \qquad \frac{}{a^{<>\forall.*}} \quad \frac{}{\tilde{a}^{<>\forall.*}} \qquad \frac{}{A \to B^{<>\forall.*}}$$

$$\frac{A_1^{<>\forall.*} \quad A_2^{<>\forall.*}}{A_1 \sqcap A_2^{<>\forall.*}} \quad \frac{A_1^{<>\forall.*}}{A_1 \sqcup A_2^{<>\forall.*}} \quad \frac{A_2^{<>\forall.*}}{A_1 \sqcup A_2^{<>\forall.*}}$$

$$\frac{\Psi \vdash \tau \quad \Psi \vdash [\tau/a]A \leqslant B \quad \Psi \vdash \forall a.A \quad B^{<>\forall.*}}{\Psi \vdash \forall a.A \leqslant B} \leqslant\forall\mathrm{L}$$

HKU PLGroup

Restriction in ∀L

A better one

$$\frac{}{\mathbb{1}^{<>\forall.*}} \quad \frac{}{\bot^{<>\forall.*}} \quad \frac{}{\top^{<>\forall.*}} \qquad \frac{}{a^{<>\forall.*}} \quad \frac{}{\tilde{a}^{<>\forall.*}} \qquad \frac{}{A \to B^{<>\forall.*}}$$

$$\frac{A_1^{<>\forall.*} \quad A_2^{<>\forall.*}}{A_1 \sqcap A_2^{<>\forall.*}} \quad \frac{A_1^{<>\forall.*}}{A_1 \sqcup A_2^{<>\forall.*}} \quad \frac{A_2^{<>\forall.*}}{A_1 \sqcup A_2^{<>\forall.*}}$$

$$\frac{\Psi \vdash \tau \quad \Psi \vdash [\tau/a]A \leqslant B \quad \Psi \vdash \forall a.A \quad B^{<>\forall.*}}{\Psi \vdash \forall a.A \leqslant B} \leqslant\forall\mathrm{L}$$

$$\forall a.(a \to \texttt{Int}) \sqcup (a \to \texttt{Bool}) \leqslant (\texttt{Int} \to \texttt{Int}) \sqcup (\texttt{Int} \to \texttt{Bool})$$

HKU PLGroup

Checking and Inference

$$\frac{\Psi \vdash e \Rightarrow A \quad \Psi \vdash A \leqslant B}{\Psi \vdash e \Leftarrow B} \Leftarrow\texttt{Sub} \qquad \frac{\Psi, x : A \vdash e \Leftarrow B}{\Psi \vdash \lambda x.\ e \Leftarrow A \to B} \Leftarrow\to \qquad \frac{\Psi, x : \bot \vdash e \Leftarrow \top}{\Psi \vdash \lambda x.\ e \Leftarrow \top} \Leftarrow\to\top$$

$$\frac{\Psi \vdash e \Leftarrow A \quad \Psi \vdash e \Leftarrow B}{\Psi \vdash e \Leftarrow A \sqcap B} \Leftarrow\sqcap \qquad \frac{\Psi \vdash e \Leftarrow A \quad \Psi \vdash B}{\Psi \vdash e \Leftarrow A \sqcup B} \Leftarrow\sqcup\texttt{L} \qquad \frac{\Psi \vdash e \Leftarrow B \quad \Psi \vdash A}{\Psi \vdash e \Leftarrow A \sqcup B} \Leftarrow\sqcup\texttt{R}$$

$$\frac{(x : A) \in \Psi}{\Psi \vdash x \Rightarrow A} \Rightarrow\texttt{Var} \qquad \frac{\Psi \vdash e \Leftarrow A}{\Psi \vdash (e : A) \Rightarrow A} \Rightarrow\texttt{Anno} \qquad \frac{\Psi, a \vdash e \Leftarrow A \quad \Psi \vdash \forall a.A}{\Psi \vdash (\Lambda a.e : A) \Rightarrow \forall a.A} \Rightarrow \qquad \frac{}{\Psi \vdash (\!|\,|\!) \Rightarrow \mathbb{1}} \Rightarrow\mathbb{1}$$

$$\frac{\Psi, x : \tau_1 \vdash e \Leftarrow \tau_2}{\Psi \vdash \lambda x.e \Rightarrow \tau_1 \to \tau_2} \Rightarrow\texttt{Mono}$$

$$\frac{\Psi \vdash e_1 \Rightarrow A \quad \Psi \vdash A \rhd B \to C \quad \Psi \vdash e_2 \Leftarrow B}{\Psi \vdash e_1\ e_2 \Rightarrow C} \Rightarrow\texttt{App} \qquad \frac{\Psi \vdash e \Rightarrow A \quad \Psi \vdash A \circ B \Rrightarrow C \quad \Psi \vdash B}{\Psi \vdash e\ @B \Rightarrow C} \Rightarrow\texttt{TApp}$$

A Glimpse at the Algorithmic System

$$\text{Algorithmic worklist} \quad \Gamma ::= \quad \cdot \mid \Gamma, a : \square \mid \Gamma, a : \tilde{\square} \mid \Gamma, \widehat{\alpha} \mid x : A \mid \Gamma \Vdash w$$

- Algorithmically finding instantiation
  - Generate existential variables for types to solve

$$\Gamma \Vdash \forall a.\ A \leqslant C \longrightarrow \Gamma, \widehat{\alpha} \Vdash [\widehat{\alpha}/a]A \leqslant C$$

  - Solve them when we find a solution

$$\Gamma \Vdash \widehat{\alpha} \leqslant \tau \longrightarrow \{\tau/\widehat{\alpha}\}\Gamma$$
$$\Gamma \Vdash \tau \leqslant \widehat{\alpha} \longrightarrow \{\tau/\widehat{\alpha}\}\Gamma$$

HKU PLGroup

A Glimpse at the Algorithmic System

$$\text{Algorithmic worklist} \quad \Gamma ::= \quad \cdot \mid \Gamma, a : \square \mid \Gamma, a : \tilde{\square} \mid \Gamma, \widehat{\alpha} \mid x : A \mid \Gamma \Vdash w$$

- Algorithmically finding instantiation
  - Generate existential variables for types to solve

$$\Gamma \Vdash \forall a.\ A \leqslant C \longrightarrow \Gamma, \widehat{\alpha} \Vdash [\widehat{\alpha}/a]A \leqslant C$$

  - Solve them when we find a solution

$$\Gamma \Vdash \widehat{\alpha} \leqslant \tau \longrightarrow \{\tau/\widehat{\alpha}\}\Gamma$$
$$\Gamma \Vdash \tau \leqslant \widehat{\alpha} \longrightarrow \{\tau/\widehat{\alpha}\}\Gamma$$

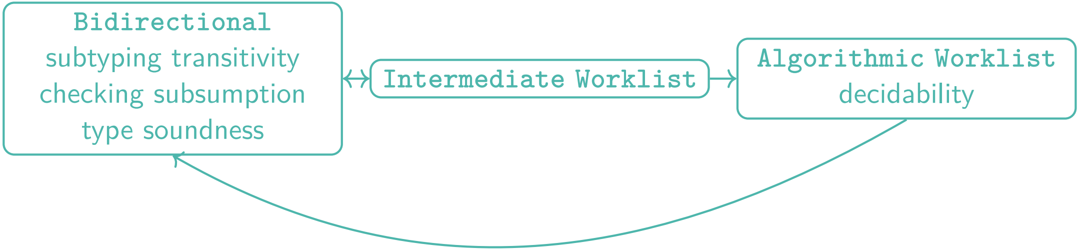
- Continuation-passing style

$$\Gamma \Vdash e \Leftarrow B \longrightarrow \Gamma \Vdash e \Rightarrow \_ \leqslant B$$

HKU PLGroup

Table of Contents

Proof Structure



The proof is done in Rocq, based on a new infrastructure using the locally nameless representation (Ott + LNGen).

Properties, Formally

Subtyping Transitivity and Checking Subsumption

Theorem (Subtyping Transitivity)

If $\Psi \vdash A \leqslant B$ *and* $\Psi \vdash B \leqslant C$ *then* $\Psi \vdash A \leqslant C$

Theorem (Checking Subsumption)

If $\Psi \vdash e \Leftarrow A$ *and* $\Psi \vdash A \leqslant B$ *then* $\Psi \vdash e \Leftarrow B$.

Theorem (Type Soundness)

If $\Psi \vdash e \Leftrightarrow A$, *then* $[\![\Psi]\!] \vdash^{f} [\![e]\!] : [\![A]\!]$

Properties, Formally

Soundness and Completeness

Theorem (Soundness)

If $\cdot \vdash e$ *and* $(\cdot \Vdash e \Rightarrow \_) \longrightarrow^*_{aw} \cdot$, *then* $(\cdot \Vdash e \Rightarrow \_) \longrightarrow^*_{d} \cdot$.

Theorem (Completeness)

If $\cdot \vdash e$ *and* $(\cdot \Vdash e \Rightarrow \_) \longrightarrow^*_{iw} \cdot$, *then* $(\cdot \Vdash e \Rightarrow \_) \longrightarrow^*_{aw} \cdot$.

Theorem (Decidability)

If $\cdot \vdash e$, *it is decidable whether* $(\cdot \Vdash e \Rightarrow \_) \longrightarrow^*_{aw} \cdot$.

HKU PLGroup

Transfer

Old Solution

Generalized completeness: If $\vdash \Gamma, \Gamma \rightsquigarrow \Omega$ and $\Gamma \longrightarrow^*_{aw} \cdot$, then $\Omega \longrightarrow^*_{dw} \cdot$.

$$\frac{}{\Omega \rightsquigarrow \Omega} \rightsquigarrow \Omega \qquad \frac{\Omega \vdash \tau \quad \Omega, [\tau/\widehat{\alpha}]\Gamma \rightsquigarrow \Omega}{\Omega, \widehat{\alpha}, \Gamma \rightsquigarrow \Omega} \rightsquigarrow \widehat{\alpha}$$

Relate intermediate worklist with (a set of) algorithmic worklists for soundness and completeness proof by interpreting free existential variables

Transfer

Old Solution

Generalized completeness: If $\vdash \Gamma, \Gamma \rightsquigarrow \Omega$ and $\Gamma \longrightarrow^*_{aw} \cdot$, then $\Omega \longrightarrow^*_{dw} \cdot$.

$$\frac{}{\Omega \rightsquigarrow \Omega} \rightsquigarrow \Omega \qquad \frac{\Omega \vdash \tau \quad \Omega, [\tau/\hat{\alpha}]\Gamma \rightsquigarrow \Omega}{\Omega, \hat{\alpha}, \Gamma \rightsquigarrow \Omega} \rightsquigarrow \hat{\alpha}$$

- Reversed definition compared to the natural definition of list;
- Proof burden of inversion lemmas to relate Γ and Ω

```
Theorem tex_all_matchL : forall E Jo A B C a b,
    tex (j (subty (all A C) B) :: E) (j (subty a b) :: Jo) -> exists a1 c1, a = all a1 c1.
```

- Substituting too eagerly breaks the structure of the algorithmic worklist and complicates proofs related to worklist substitution

Transfer

Syntax-directed Transfer

$\theta ::= \cdot \mid \theta, a \mid \theta, \tilde{a} \mid \theta, \widehat{\alpha} : \tau$

- $\theta \Vdash A^a \rightsquigarrow A^d$
- $\theta \Vdash e^a \rightsquigarrow e^d$
- $\theta \Vdash c^a \rightsquigarrow c^d$
- $\theta \Vdash w^a \rightsquigarrow w^d$
- $\theta \Vdash \Gamma \rightsquigarrow \Omega \dashv\vert \theta'$

```
Definition transfer (Ω : iworklist) (Γ : aworklist) : Prop :=
  exists θ, trans_worklist · Ω Γ θ.
```

Transfer

Syntax-directed Transfer

$\theta ::= \cdot \mid \theta, a \mid \theta, \tilde{a} \mid \theta, \widehat{\alpha} : \tau$

- $\theta \Vdash A^a \rightsquigarrow A^d$
- $\theta \Vdash e^a \rightsquigarrow e^d$
- $\theta \Vdash c^a \rightsquigarrow c^d$
- $\theta \Vdash w^a \rightsquigarrow w^d$
- $\theta \Vdash \Gamma \rightsquigarrow \Omega \dashv \theta'$

$$\theta \vDash a \rightsquigarrow a \qquad \theta \vDash \tilde{a} \rightsquigarrow \tilde{a} \qquad \theta \vDash \widehat{\alpha} \rightsquigarrow \tau$$

$$\theta \vDash \mathbb{1} \rightsquigarrow \mathbb{1} \qquad \theta \vDash \bot \rightsquigarrow \bot \qquad \theta \vDash \top \rightsquigarrow \top$$

$$\frac{\theta \Vdash A \rightsquigarrow A' \quad \theta \Vdash B \rightsquigarrow B'}{\theta \vDash A \to B \rightsquigarrow A' \to B'} \qquad \frac{\theta, a \Vdash A \rightsquigarrow A'}{\theta \vDash \forall a.\ A \rightsquigarrow \forall a.\ A'}$$

$$\frac{\theta \Vdash A \rightsquigarrow A' \quad \theta \Vdash B \rightsquigarrow B'}{\theta \vDash A \sqcap B \rightsquigarrow A' \sqcap B'} \qquad \frac{\theta \Vdash A \rightsquigarrow A' \quad \theta \Vdash B \rightsquigarrow B'}{\theta \vDash A \sqcup B \rightsquigarrow A' \sqcup B'}$$

```
Definition transfer (Ω : iworklist) (Γ : aworklist) : Prop :=
  exists θ, trans_worklist · Ω Γ θ.
```

HKU PLGroup

Transfer

Syntax-directed Transfer

$\theta ::= \cdot \mid \theta, a \mid \theta, \tilde{a} \mid \theta, \widehat{\alpha} : \tau$

- $\theta \Vdash A^a \rightsquigarrow A^d$
- $\theta \Vdash e^a \rightsquigarrow e^d$
- $\theta \Vdash c^a \rightsquigarrow c^d$
- $\theta \Vdash w^a \rightsquigarrow w^d$
- $\theta \Vdash \Gamma \rightsquigarrow \Omega \dashv \theta'$

$$\theta \Vdash \cdot \rightsquigarrow \cdot \dashv \theta'$$

$$\frac{\theta \Vdash \Gamma \rightsquigarrow \Omega \dashv \theta'}{\theta \Vdash \Gamma, a \rightsquigarrow \Omega, a \dashv \theta', a} \qquad \frac{\theta \Vdash \Gamma \rightsquigarrow \Omega \dashv \theta'}{\theta \Vdash \Gamma, \tilde{a} \rightsquigarrow \Omega, \tilde{a} \dashv \theta', \tilde{a}}$$

$$\frac{\theta \Vdash \Gamma \rightsquigarrow \Omega \dashv \theta' \quad \theta' \Vdash w^a \rightsquigarrow w^d}{\theta \Vdash \Gamma \vDash w^a \rightsquigarrow \Omega \vDash w^d \dashv \theta'} \qquad \frac{\theta \Vdash \Gamma \rightsquigarrow \Omega \dashv \theta' \quad \theta' \Vdash A^a \rightsquigarrow A^d}{\theta \Vdash \Gamma, x : A^a \rightsquigarrow \Omega, x : A^d \dashv \theta'}$$

$$\frac{\theta \Vdash \Gamma \rightsquigarrow \Omega \dashv \theta'}{\theta \Vdash \Gamma, \widehat{\alpha} \rightsquigarrow \Omega \dashv \theta', \widehat{\alpha} : \tau}$$

```
Definition transfer (Ω : iworklist) (Γ : aworklist) : Prop :=
  exists θ, trans_worklist · Ω Γ θ.
```

Continuation Passing Style

Defunctionalization

$$\Gamma \Vdash 1 \Rightarrow (\_ \leqslant \texttt{Int}) \longrightarrow \Gamma \Vdash (\_ \leqslant \texttt{Int}) \diamond \texttt{Int} \longrightarrow \Gamma \Vdash \texttt{Int} \leqslant \texttt{Int}$$

```
Inductive cont : Set :=
| cont_sub : typ -> cont
...

Inductive work : Set :=
| work_sub : typ -> typ -> work
| work_apply : cont -> typ -> work
...

Inductive apply_cont : cont -> typ -> work -> Prop :=
| apply_cont_sub : apply_cont (cont_sub B) A (work_sub A B)
...
```

Continuation Passing Style

Old Solution

When e=1 expression infers the A=Int type, check if it is a subtype of Int

- $\Gamma \Vdash 1 \Rightarrow (\texttt{fun}\ \ t : t \leqslant 1) \longleftarrow \Gamma \Vdash (\texttt{fun}\ \ t : t \leqslant \texttt{Int})\ 1$

  ```
  Inductive work : Set :=
  | work_infer : exp -> (typ -> work) -> work
  ```

  HOAS cannot be encoded in Ott directly, because it exploits features of meta-language (i.e. Coq)

- $\Gamma \Vdash 1 \Rightarrow_a (a \leqslant \texttt{Int}) \longrightarrow \Gamma \Vdash \{\texttt{Int}/a\}(a \leqslant \texttt{Int})$

  ```
  Inductive work : Set :=
  | work_infer : exp -> work -> work
  ```

  LNgen has poor support for multiple binders, which is required by the matching relation

HKU PLGroup

Table of Contents

Contribution

- A bidirectional type system for higher-rank polymorphism and intersection/union type
  - Predicative implicit instantiation
  - Impredicative explicit type applications
  - Subtyping transitivity, checking subsumption, type safety
- A sound, complete and decidable algorithm for the base system
  - Worklist formulation
- A sound and complete algorithm for the system with record extension
- A sound algorithm for the system with record extension and relaxed monotype definition
- Mechanical formalization and implementation
  - New proof infrastructure based on LN
  - All theorems are verified in Coq (LOC: 40000+ for the base system)
  - Haskell implementation

Bibliography I

- Castagna, Giuseppe et al. (2024). "Polymorphic Type Inference for Dynamic Languages". In: *Proc. ACM Program. Lang.* 8.POPL.
- Dudenhefner, Andrej et al. (2016). "The Intersection Type Unification Problem". In: vol. 52. Dagstuhl, Germany. ISBN: 978-3-95977-010-1. DOI: 10.4230/LIPIcs.FSCD.2016.19.
- Dunfield, Jana et al. (2013). "Complete and Easy Bidirectional Typechecking for Higher-rank Polymorphism". In: *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming*. ICFP, pp. 429–442. DOI: 10.1145/2500365.2500582.
- Zhao, Jinxu et al. (2022). "Elementary Type Inference". In: *36th European Conference on Object-Oriented Programming (ECOOP 2022)*. Vol. 222. Leibniz International Proceedings in Informatics (LIPIcs), 2:1–2:28. ISBN: 978-3-95977-225-9.

A Simple Extension To Encode Record

Label names	l		
Declarative types	A, B, C	$::=$	$\ldots \mid \texttt{Label}\ l$
Monotypes	τ	$::=$	$\ldots \mid \texttt{Label}\ l$
Expressions	e	$::=$	$\ldots \mid \langle l \mapsto e \rangle \mid \langle l_1 \mapsto e_1, e_2 \rangle \mid e.l$

$$\frac{}{\Psi \vdash \texttt{Label}\ l \leqslant \texttt{Label}\ l} \leqslant\texttt{Label}$$

$$\frac{\Psi \vdash e \Rightarrow A}{\Psi \vdash \langle l \mapsto e \rangle \Rightarrow \texttt{Label}\ l \to A} \Rightarrow\langle\rangle \qquad \frac{\Psi \vdash e_1 \Rightarrow A_1 \quad \Psi \vdash e_2 \Rightarrow A_2}{\Psi \vdash \langle l_1 \mapsto e_1, e_2 \rangle \Rightarrow (\texttt{Label}\ l_1 \to A_1) \sqcap A_2} \Rightarrow\langle\rangle\texttt{Cons}$$

$$\frac{\Psi \vdash e \Rightarrow A \quad \Psi \vdash A \rhd B \to C \quad \Psi \vdash \texttt{Label}\ l \leqslant B}{\Psi \vdash e.l \Rightarrow C} \Rightarrow\langle\rangle\texttt{Proj}$$

HKU PLGroup

Discussion

Why not instantiate to complex types directly?

$$
\begin{array}{rcll}
\text{Monotypes} & \tau & ::= & 1 \mid a \mid \tau_1 \rightarrow \tau_2 \mid \tau_1 \sqcup \tau_2 \mid \tau_1 \sqcap \tau_2 \\
\text{Bound types} & \sigma & ::= & \bot \mid \top \mid \tau \\
\text{Algorithmic Worklist} & \Gamma & ::= & \cdot \mid \Gamma, a \mid \Gamma, \tilde{a} \mid \Gamma, \sigma_1 < \widehat{\alpha} < \sigma_2
\end{array}
$$

$$
\Gamma[\sigma_1 < \widehat{\alpha} < \sigma_2] \Vdash \widehat{\alpha} \leqslant \tau \longrightarrow \ ? \qquad \text{when } \widehat{\alpha} \in \mathrm{fv}(\tau)
$$

Discussion

Why not instantiate to complex types directly?

$$
\begin{array}{rccl}
\text{Monotypes} & \tau & ::= & 1 \mid a \mid \tau_1 \to \tau_2 \mid \tau_1 \sqcup \tau_2 \mid \tau_1 \sqcap \tau_2 \\
\text{Bound types} & \sigma & ::= & \bot \mid \top \mid \tau \\
\text{Algorithmic Worklist} & \Gamma & ::= & \cdot \mid \Gamma, a \mid \Gamma, \tilde{a} \mid \Gamma, \sigma_1 < \hat{\alpha} < \sigma_2
\end{array}
$$

Definition (Subtyping Satisfiability with Intersection Types)

Given a set of constraints $C = \{\sigma_1 \leqslant \tau_1, \ldots, \sigma_n \leqslant \tau_n\}$, is there a substitution $S : \mathbb{V} \to \mathbb{T}$ such that $S(\sigma_i) \leqslant S(\tau_i), \forall i \in \{1, \ldots, n\}$?

Theorem (Hardness of Intersection Type Satisfiability (Dudenhefner et al. 2016))

The intersection type satisfiability is at best EXPTIME*-hard, if decidable.*

HKU PLGroup