Bidirectional Type Checking for Existential Types with Higher-Rank Polymorphism

Who? Hasti Toossi, Ningning Xie

From? University of Toronto

When? ESOP 2026

Existential types

Types

$Vec\ n\ a$

Example

$$filter :: \forall n\ a.\ (a \to Bool) \to Vec\ n\ a \to \exists m.\ Vec\ m\ a$$

$$filter = \lambda p\ vec \to \textbf{case}\ vec\ \textbf{of}$$

$$Nil \to Nil$$

$$(:>)\ x\ xs$$

$$\mid p\ x \to (:>)\ x\ (filter\ p\ xs)$$

$$\mid otherwise \to filter\ p\ xs$$

(Weak) Existential types

Types

$$pack\ t, e\ as\ \exists a.\ t'$$
$$unpack\ e\ as\ t, x\ \textbf{in}\ e'$$

Example

$$filter :: \forall n\ a.\ (a \to Bool) \to Vec\ n\ a \to \exists m.\ Vec\ m\ a$$
$$filter = \Lambda n\ a.\ \lambda(p :: a \to Bool)\ (vec :: Vec\ n\ a) \to$$
$$\quad \textbf{case}\ vec\ \textbf{of}$$
$$\quad\quad Nil \to pack\ Zero, Nil\ as\ \exists m.\ Vec\ m\ a$$
$$\quad\quad (:>)\ n\ a\ (x :: a)\ (xs :: Vec\ n\ a)$$
$$\quad\quad\quad \mid p\ x \to unpack\ (filter\ n\ a\ p\ xs)\ as\ n1, ys\ \textbf{in}$$
$$\quad\quad\quad\quad pack\ Succ\ n1, (:>)\ n1\ a\ x\ ys\ as\ \exists m.\ Vec\ m\ a$$
$$\quad\quad\quad \mid otherwise \to filter\ n\ a\ p\ xs$$

Strong Existential types

Types

$pack\ t, e\ as\ \exists a.\ t'$

$\lfloor e \rfloor$

Expressions

$open\ e$

Example

$$
\begin{aligned}
&filter :: \forall n\ a.\ (a \to Bool) \to Vec\ n\ a \to \exists m.\ Vec\ m\ a \\
&filter = \Lambda n\ a.\ \lambda(p :: a \to Bool)\ (vec :: Vec\ n\ a) \to \\
&\quad \textbf{case}\ vec\ \textbf{of} \\
&\qquad Nil \to pack\ Zero, Nil\ as\ \exists m.\ Vec\ m\ a \\
&\qquad (:>)\ n\ a\ (x :: a)\ (xs :: Vec\ n\ a) \\
&\qquad\quad \mid p\ x \to \textbf{let}\ ys = filter\ n\ a\ p\ xs\ \textbf{in} \\
&\qquad\qquad pack\ Succ\ \lfloor ys \rfloor, (:>)\ \lfloor ys \rfloor\ a\ x\ (open\ ys) \\
&\qquad\qquad\quad as\ \exists m.\ Vec\ m\ a \\
&\qquad\quad \mid otherwise \to filter\ n1\ a\ p\ xs
\end{aligned}
$$

Existential types

Types

$Vec\ n\ a$

Example

$$filter :: \forall n\ a.\ (a \to Bool) \to Vec\ n\ a \to \exists m.\ Vec\ m\ a$$
$$filter = \lambda p\ vec \to \textbf{case}\ vec\ \textbf{of}$$
$$\quad Nil \to Nil$$
$$\quad (:>)\ x\ xs$$
$$\qquad \mid p\ x \to (:>)\ x\ (filter\ p\ xs)$$
$$\qquad \mid otherwise \to filter\ p\ xs$$

Prior Work

An Existential Crisis Resolved

Type Inference for First-Class Existential Types

RICHARD A. EISENBERG, Tweag, France
GUILLAUME DUBOC, ENS Lyon, France and Tweag, France
STEPHANIE WEIRICH, University of Pennsylvania, USA
DANIEL LEE, University of Pennsylvania, USA

Prior Work

EDWL

- A declarative type system
  - strong existential types
  - higher-rank polymorphism
- An elaboration of the decl. system
  - type soundness

Prior Work

EDWL
ICFP 2021

An Existential Crisis Resolved

Type Inference for First-Class Existential Types

RICHARD A. EISENBERG, Tweag, France
GUILLAUME DUBOC, ENS Lyon, France and Tweag, France
STEPHANIE WEIRICH, University of Pennsylvania, USA
DANIEL LEE, University of Pennsylvania, USA

Quote

"A real implementation might use unification variables, but we here rely on the rich body of literature [e.g. Dunfield and Krishnaswami 2013] that allows us to guess monotypes during type inference, knowing how to translate this convention into an implementation using unification variables."

Contributions

This Work

- A declarative type system
  - strong existential types
  - higher-rank polymorphism
  - subtyping

Contributions

This Work

- A declarative type system
  - strong existential types
  - higher-rank polymorphism
  - subtyping
- An algorithmic type system
  - unification and promotion
  - sound but incomplete w.r.t. the decl. system
  - complete w.r.t. 2 fragments of the decl. system

Contributions

This Work

- A declarative type system
  - strong existential types
  - higher-rank polymorphism
  - subtyping
- An algorithmic type system
  - unification and promotion
  - sound but incomplete w.r.t. the decl. system
  - complete w.r.t. 2 fragments of the decl. system
- An elaboration of the decl. system
  - type soundness

Contributions

This Work

- A declarative type system
  - strong existential types
  - higher-rank polymorphism
  - subtyping
- An algorithmic type system
  - unification and promotion
  - sound but incomplete w.r.t. the decl. system
  - complete w.r.t. 2 fragments of the decl. system
- An elaboration of the decl. system
  - type soundness
- Haskell implementation

Declarative System

Types

$\exists\, a.\epsilon$

$\lfloor e : \exists\, a.\epsilon \rfloor$

Key Rule

D-I-Abs

$$\frac{\Gamma, x : \tau \vdash e \Rightarrow \sigma \qquad \Gamma, x : \tau \vdash \sigma \rightsquigarrow_{\forall} \epsilon \qquad \overline{a} \text{ fresh} \qquad \epsilon' = [\overline{a}/\lfloor \epsilon \rfloor_x]\epsilon}{\Gamma \vdash \lambda x.\ e \Rightarrow \tau \to \exists\, \overline{a}.\epsilon'}$$

Inferring Existentials and Projections

Example

- $\vdash \lambda x : \mathsf{Int}.\ \lambda y : \lfloor x \rfloor.\ y \Rightarrow ?$

Inferring Existentials and Projections

Example

- $\bullet \vdash \lambda x : \mathsf{Int}.\ \lambda y : \lfloor x \rfloor.\ y \Rightarrow ?$
- 1 $x : \mathsf{Int} \vdash \lambda y : \lfloor x \rfloor.\ y \Rightarrow \lfloor x \rfloor \to \lfloor x \rfloor$

Inferring Existentials and Projections

Example

- $\bullet \vdash \lambda x : \mathsf{Int}.\ \lambda y : \lfloor x \rfloor.\ y \Rightarrow \mathsf{Int} \to \lfloor x \rfloor \Rightarrow \lfloor x \rfloor$
- 1 $x : \mathsf{Int} \vdash \lambda y : \lfloor x \rfloor.\ y \Rightarrow \lfloor x \rfloor \to \lfloor x \rfloor$

Inferring Existentials and Projections

Example

- $\vdash \lambda x : \mathsf{Int}.\ \lambda y : \lfloor x \rfloor.\ y \Rightarrow \mathsf{Int} \to \exists\, a.a \to a$

1. $x : \mathsf{Int} \vdash \lambda y : \lfloor x \rfloor.\ y \Rightarrow \lfloor x \rfloor \to \lfloor x \rfloor$

Inferring Existentials and Projections

Example

- $\bullet \vdash \lambda x : \mathsf{Int}.\ \lambda y : \lfloor x \rfloor.\ y \Rightarrow \mathsf{Int} \to \exists\, a.a \to a$
- $\boxed{1}\ x : \mathsf{Int} \vdash \lambda y : \lfloor x \rfloor.\ y \Rightarrow \lfloor x \rfloor \to \lfloor x \rfloor$

Example

Let $e = \lambda x : \mathsf{Int}.\ \lambda y : \lfloor x \rfloor.\ y$.

- $\bullet \vdash e\, 1\, e' \Rightarrow\ ?$

Inferring Existentials and Projections

Example

- $\bullet \vdash \lambda x : \mathsf{Int}.\ \lambda y : \lfloor x \rfloor.\ y \Rightarrow \mathsf{Int} \to \exists\, a.a \to a$

1. $x : \mathsf{Int} \vdash \lambda y : \lfloor x \rfloor.\ y \Rightarrow \lfloor x \rfloor \to \lfloor x \rfloor$

Example

Let $e = \lambda x : \mathsf{Int}.\ \lambda y : \lfloor x \rfloor.\ y$.

- $\bullet \vdash e\, 1\, e' \Rightarrow ?$

1. $\bullet \vdash e\, 1 \Rightarrow \exists\, a.a \to a$

Inferring Existentials and Projections

Example

- $\bullet \vdash \lambda x : \mathsf{Int}.\ \lambda y : \lfloor x \rfloor.\ y \Rightarrow \mathsf{Int} \to \exists\, a.a \to a$

1. $x : \mathsf{Int} \vdash \lambda y : \lfloor x \rfloor.\ y \Rightarrow \lfloor x \rfloor \to \lfloor x \rfloor$

Example

Let $e = \lambda x : \mathsf{Int}.\ \lambda y : \lfloor x \rfloor.\ y$.

- $\bullet \vdash e\, 1\, e' \Rightarrow ?$

1. $\bullet \vdash e\, 1 \Rightarrow \exists\, a.a \to a$
2. Instantiate $\exists\, a.a \to a$ to $\lfloor e\, 1 : \exists\, a.a \to a \rfloor \to \lfloor e\, 1 : \exists\, a.a \to a \rfloor$.

Inferring Existentials and Projections

Example

- $\bullet \vdash \lambda x : \mathsf{Int}.\ \lambda y : \lfloor x \rfloor.\ y \Rightarrow \mathsf{Int} \to \exists\, a.a \to a$
1. $x : \mathsf{Int} \vdash \lambda y : \lfloor x \rfloor.\ y \Rightarrow \lfloor x \rfloor \to \lfloor x \rfloor$

Example

Let $e = \lambda x : \mathsf{Int}.\ \lambda y : \lfloor x \rfloor.\ y$.

- $\bullet \vdash e\,1\,e' \Rightarrow \lfloor e\,1 : \exists\, a.a \to a \rfloor$
1. $\bullet \vdash e\,1 \Rightarrow \exists\, a.a \to a$
2. Instantiate $\exists\, a.a \to a$ to $\lfloor e\,1 : \exists\, a.a \to a \rfloor \to \lfloor e\,1 : \exists\, a.a \to a \rfloor$.
3. $\bullet \vdash e' \Leftarrow \lfloor e\,1 : \exists\, a.a \to a \rfloor$

Prior Work

DK
ICFP 2013

Complete and Easy Bidirectional Typechecking for Higher-Rank Polymorphism

Jana Dunfield Neelakantan R. Krishnaswami
Max Planck Institute for Software Systems
Kaiserslautern and Saarbrücken, Germany
jd169@queensu.ca nk480@cl.cam.ac.uk

Algorithmic System

Types	$\hat{a}$
Contexts	$\Delta, \hat{a}$ $\Delta, \hat{a} = \tau$
Judgements	$\Delta_1 \vdash e \Rightarrow \sigma \dashv \Delta_2$

Declarative vs. Algorithmic Inference

Example

- $\bullet \vdash \lambda x : \mathsf{Int}.\ \lambda y.\ y \Rightarrow ?$
- 1 $x : \mathsf{Int} \vdash \lambda y.\ y \Rightarrow ?$

- $\bullet \vdash \lambda x : \mathsf{Int}.\ \lambda y.\ y \Rightarrow ?$
- 1 $x : \mathsf{Int} \vdash \lambda y.\ y \Rightarrow ?$

Declarative vs. Algorithmic Inference

Example

- $\bullet \vdash \lambda x : \mathsf{Int}.\ \lambda y.\ y \Rightarrow \mathsf{Int} \to \hat{a} \to \hat{a} \dashv \hat{a}$
- 1 $x : \mathsf{Int} \vdash \lambda y.\ y \Rightarrow \hat{a} \to \hat{a} \dashv \hat{a}$

- $\bullet \vdash \lambda x : \mathsf{Int}.\ \lambda y.\ y \Rightarrow ?$
- 1 $x : \mathsf{Int} \vdash \lambda y.\ y \Rightarrow \tau \to \tau$

Declarative vs. Algorithmic Inference

Example

- $\bullet \vdash \lambda x : \mathsf{Int}.\ \lambda y.\ y \Rightarrow \mathsf{Int} \to \hat{a} \to \hat{a} \dashv \hat{a}$
- $1\ x : \mathsf{Int} \vdash \lambda y.\ y \Rightarrow \hat{a} \to \hat{a} \dashv \hat{a}$

- $\bullet \vdash \lambda x : \mathsf{Int}.\ \lambda y.\ y \Rightarrow ?$
- $1\ x : \mathsf{Int} \vdash \lambda y.\ y \Rightarrow \tau \to \tau$

- $\bullet \vdash \lambda x : \mathsf{Int}.\ \lambda y.\ y \Rightarrow \mathsf{Int} \to \mathsf{Int} \to \mathsf{Int}$
- $1\ x : \mathsf{Int} \vdash \lambda y.\ y \Rightarrow \mathsf{Int} \to \mathsf{Int}$

- $\bullet \vdash \lambda x : \mathsf{Int}.\ \lambda y.\ y \Rightarrow \mathsf{Int} \to (\mathsf{Int} \to \mathsf{Int}) \to (\mathsf{Int} \to \mathsf{Int})$
- $1\ x : \mathsf{Int} \vdash \lambda y.\ y \Rightarrow (\mathsf{Int} \to \mathsf{Int}) \to (\mathsf{Int} \to \mathsf{Int})$

Soundness and (In)completeness

Soundness If the algorithmic system infers type σ' for e, the declarative system can infer any solution σ of σ'.

Completeness? If the declarative system can infer type σ for e, the algorithmic system infers σ' such that σ is a solution of σ'.

Declarative vs. Algorithmic Inference

Example

- $\bullet \vdash \lambda x : \mathsf{Int}.\ \lambda y.\ y \Rightarrow \mathsf{Int} \to \hat{a} \to \hat{a} \dashv \hat{a}$
- $1\ \ x : \mathsf{Int} \vdash \lambda y.\ y \Rightarrow \hat{a} \to \hat{a} \dashv \hat{a}$

- $\bullet \vdash \lambda x : \mathsf{Int}.\ \lambda y.\ y \Rightarrow ?$
- $1\ \ x : \mathsf{Int} \vdash \lambda y.\ y \Rightarrow \tau \to \tau$

- $\bullet \vdash \lambda x : \mathsf{Int}.\ \lambda y.\ y \Rightarrow \mathsf{Int} \to \mathsf{Int} \to \mathsf{Int}$
- $1\ \ x : \mathsf{Int} \vdash \lambda y.\ y \Rightarrow \mathsf{Int} \to \mathsf{Int}$

- $\bullet \vdash \lambda x : \mathsf{Int}.\ \lambda y.\ y \Rightarrow \mathsf{Int} \to (\mathsf{Int} \to \mathsf{Int}) \to (\mathsf{Int} \to \mathsf{Int})$
- $1\ \ x : \mathsf{Int} \vdash \lambda y.\ y \Rightarrow (\mathsf{Int} \to \mathsf{Int}) \to (\mathsf{Int} \to \mathsf{Int})$

- $\bullet \vdash \lambda x : \mathsf{Int}.\ \lambda y.\ y \Rightarrow \mathsf{Int} \to \exists\, a.a \to a$
- $1\ \ x : \mathsf{Int} \vdash \lambda y.\ y \Rightarrow \lfloor x \rfloor \to \lfloor x \rfloor$

Soundness and (In)completeness

Table

Types	Is σ' a solution of σ?
$\mathsf{Int} \to \mathsf{Int} \to \mathsf{Int}$ $\mathsf{Int} \to \mathsf{Int} \to \mathsf{Int}$	yes
$\mathsf{Int} \to \exists\, a.a \to a$ $\mathsf{Int} \to \exists\, a.a \to a$	yes
$\mathsf{Int} \to \mathsf{Int} \to \mathsf{Int}$ $\mathsf{Int} \to \hat{a} \to \hat{a}$	yes
$\mathsf{Int} \to (\mathsf{Int} \to a) \to (\mathsf{Int} \to a)$ $\mathsf{Int} \to \hat{a} \to \hat{a}$	yes
$\mathsf{Int} \to \exists\, a.a \to a$ $\mathsf{Int} \to \hat{a} \to \hat{a}$	no
$\mathsf{Int} \to \exists\, a.a \to a \to a$ $\mathsf{Int} \to \exists\, b.b \to \hat{a} \to \hat{a}$	no
$\exists\, a.a \to \mathsf{Int} \to \exists\, b.b \to b$ $\exists\, a.a \to \mathsf{Int} \to \hat{b} \to \hat{b}$	no

Soundness and (In)completeness

Soundness If the algorithmic system infers type σ' for e, the declarative system can infer any solution σ of σ'.

Completeness? If the declarative system can infer type σ for e, the algorithmic system infers σ' compatible with σ.

Soundness and (In)completeness

Table

Types	solution	compatible
$\mathsf{Int} \to \mathsf{Int} \to \mathsf{Int}$ $\mathsf{Int} \to \mathsf{Int} \to \mathsf{Int}$	yes	yes
$\mathsf{Int} \to \exists\, a.a \to a$ $\mathsf{Int} \to \exists\, a.a \to a$	yes	yes
$\mathsf{Int} \to \mathsf{Int} \to \mathsf{Int}$ $\mathsf{Int} \to \hat{a} \to \hat{a}$	yes	yes
$\mathsf{Int} \to (\mathsf{Int} \to a) \to (\mathsf{Int} \to a)$ $\mathsf{Int} \to \hat{a} \to \hat{a}$	yes	yes
$\mathsf{Int} \to \exists\, a.a \to a$ $\mathsf{Int} \to \hat{a} \to \hat{a}$	no	yes
$\mathsf{Int} \to \exists\, a.a \to a \to a$ $\mathsf{Int} \to \exists\, b.b \to \hat{a} \to \hat{a}$	no	yes
$\exists\, a.a \to \mathsf{Int} \to \exists\, b.b \to b$ $\exists\, a.a \to \mathsf{Int} \to \hat{b} \to \hat{b}$	no	yes
$\lfloor e : \exists\, a.a \to \mathsf{Int} \to \exists\, b.b \to b \rfloor$ $\lfloor e : \exists\, a.a \to \mathsf{Int} \to \hat{b} \to \hat{b} \rfloor$	no	no

Restricted Inference

Table

Types	solution	compatible
$\mathsf{Int} \to \mathsf{Int} \to \mathsf{Int}$ $\mathsf{Int} \to \mathsf{Int} \to \mathsf{Int}$	yes	yes
$\mathsf{Int} \to \exists\, a.a \to a$ $\mathsf{Int} \to \exists\, a.a \to a$	yes	yes
$\mathsf{Int} \to \mathsf{Int} \to \mathsf{Int}$ $\mathsf{Int} \to \hat{a} \to \hat{a}$	yes	yes
$\mathsf{Int} \to (\mathsf{Int} \to a) \to (\mathsf{Int} \to a)$ $\mathsf{Int} \to \hat{a} \to \hat{a}$	yes	yes
~~$\mathsf{Int} \to \exists\, a.a \to a$~~ $\mathsf{Int} \to \hat{a} \to \hat{a}$	no	yes
~~$\mathsf{Int} \to \exists\, a.a \to a \to a$~~ $\mathsf{Int} \to \exists\, b.b \to \hat{a} \to \hat{a}$	no	yes
~~$\exists\, a.a \to \mathsf{Int} \to \exists\, b.b \to b$~~ $\exists\, a.a \to \mathsf{Int} \to \hat{b} \to \hat{b}$	no	yes
~~$\lfloor e : \exists\, a.a \to \mathsf{Int} \to \exists\, b.b \to b \rfloor$~~ ~~$\lfloor e : \exists\, a.a \to \mathsf{Int} \to \hat{b} \to \hat{b} \rfloor$~~	no	no

Restricted Inference

Predicate

$$\min_{\exists}(\Gamma \vdash e \Rightarrow \sigma) \triangleq \forall \sigma'.\, \Gamma \vdash e \Rightarrow \sigma' \Longrightarrow |\exists(\sigma)| \leq |\exists(\sigma')|$$

Key Rule

D-I-Abs1

$$\frac{\begin{array}{c}\Gamma, x : \tau \vdash e \Rightarrow_1 \sigma \\ \Gamma, x : \tau \vdash \sigma \rightsquigarrow_{\forall} \epsilon \\ \bar{a} \text{ fresh} \qquad \epsilon' = [\bar{a}/\lfloor \epsilon \rfloor_x]\epsilon \\ \min_{\exists}(\Gamma \vdash \lambda x.\ e \Rightarrow \tau \to \exists \bar{a}.\epsilon')\end{array}}{\Gamma \vdash \lambda x.\ e \Rightarrow_1 \tau \to \exists \bar{a}.\epsilon'}$$

Completeness

If the restricted declarative system (1) can infer σ for e, the algorithmic system infers σ' such that σ is a solution of σ'.

Restricted Instantiation

Table

Types	solution	compatible
$\mathsf{Int} \to \mathsf{Int} \to \mathsf{Int}$ $\mathsf{Int} \to \mathsf{Int} \to \mathsf{Int}$	yes	yes
$\mathsf{Int} \to \exists\, a.a \to a$ $\mathsf{Int} \to \exists\, a.a \to a$	yes	yes
$\mathsf{Int} \to \mathsf{Int} \to \mathsf{Int}$ $\mathsf{Int} \to \hat{a} \to \hat{a}$	yes	yes
$\mathsf{Int} \to (\mathsf{Int} \to a) \to (\mathsf{Int} \to a)$ $\mathsf{Int} \to \hat{a} \to \hat{a}$	yes	yes
$\mathsf{Int} \to \exists\, a.a \to a$ $\mathsf{Int} \to \hat{a} \to \hat{a}$	no	yes
$\mathsf{Int} \to \exists\, a.a \to a \to a$ $\mathsf{Int} \to \exists\, b.b \to \hat{a} \to \hat{a}$	no	yes
$\exists\, a.a \to \mathsf{Int} \to \exists\, b.b \to b$ $\exists\, a.a \to \mathsf{Int} \to \hat{b} \to \hat{b}$	no	yes
~~$\lfloor e : \exists\, a.a \to \mathsf{Int} \to \exists\, b.b \to b \rfloor$~~ ~~$\lfloor e : \exists\, a.a \to \mathsf{Int} \to \hat{b} \to \hat{b} \rfloor$~~	no	no

Restricted Instantiation

Predicate

$$\min_{\exists}(\Gamma \vdash e \Rightarrow \sigma) \triangleq \forall \sigma'.\, \Gamma \vdash e \Rightarrow \sigma' \implies |\exists(\sigma)| \leq |\exists(\sigma')|$$

Key Rule

D-I-App2

$$\frac{\Gamma \vdash e \Rightarrow_2 \sigma \quad \min_{\exists}(\Gamma \vdash e \Rightarrow \sigma) \quad \Gamma \vdash e : \sigma \rightsquigarrow \sigma_1 \to \sigma_2 \quad \Gamma \vdash e_1 \Leftarrow_2 \sigma_1}{\Gamma \vdash e\, e_1 \Rightarrow_2 \sigma_2}$$

D-C-Sub2

$$\frac{\Gamma \vdash e \Rightarrow_2 \sigma \quad \min_{\exists}(\Gamma \vdash e \Rightarrow \sigma) \quad \Gamma \vdash e : \sigma <: \rho}{\Gamma \vdash e \Leftarrow_2 \rho}$$

Completeness

If the restricted declarative system (2) can infer σ for e, the algorithmic system infers type σ' compatible with σ.

More in the paper

This Work

- A declarative type system
  - strong existential types
  - higher-rank polymorphism
  - subtyping
- An algorithmic type system
  - unification and promotion
  - sound but incomplete w.r.t. the decl. system
  - complete w.r.t. 2 fragments of the decl. system
- An elaboration of the decl. system
  - type soundness
- Haskell implementation

More in the paper

This Work

- A declarative type system
  - strong existential types
  - higher-rank polymorphism
  - subtyping
- An algorithmic type system
  - unification and promotion
  - sound but incomplete w.r.t. the decl. system
  - complete w.r.t. 2 fragments of the decl. system
- An elaboration of the decl. system
  - type soundness
- Haskell implementation

Our paper

TX
ESOP 2026

Bidirectional Type Checking for Existential Types with Higher-Rank Polymorphism

Hasti Toossi and Ningning Xie

University of Toronto, Toronto, Canada
{hasti2c,ningningxie}@cs.toronto.edu

Abstract. This paper presents a new type system that combines strong existential types, higher-rank polymorphism, and polymorphic subtyping. We begin by presenting a declarative type system, and contributing several metatheoretic results. We then detail an algorithmic type inference system featuring bidirectional type checking. We prove that the algorithm is sound. Moreover, we propose two design variants of the declarative system with respect to which the algorithmic system is complete. This work can serve as a basis for further studies on strong existentials, their corresponding type inference algorithms, and their combinations with higher-rank polymorphism and subtyping.