Polymorphism with Typed Holes

Presenters: Adam Chen (Stevens Institute of Technology), Thomas Porter
Non-presenting Author: Cyrus Omar (University of Michigan)

Trends in Functional Programming (TFP)
10 January 2024

Introduction

- Live programming
  - Continuous feedback to programmer.
  - Must address semantics of incomplete programs.
  - Gradual Typing
- History of gradual typing and polymorphism
  - Fulfilling both parametricity and the gradual guarantee is hard.
  - Igarashi et al 2017, New et al 2020, Labrada et al 2022

The System

- Based off the system in Hazelnut Live (PACMPL 2019)
- Core feature: Typed Holes
    - During elaboration, store what type each hole needs to be in a “hole context”.

```
let x = 2 + ⬡ in
let y = 5 in 3 * x + 2 * y

≡ 3 * (2 + ?) + 10
```

The System

- User-facing gradually typed calculus
  - Bidirectionally Typed
- Internal cast calculus
  - Terms reduced during execution.

- Add rules for type functions to both.

The System

- Add bidirectional typing rules:

$\boxed{\Sigma;\Gamma \vdash e \Rightarrow \tau}$ Expression e synthesizes type τ in context $\Sigma;\Gamma$

$$\text{STLAM}\quad \frac{\Sigma,\alpha;\Gamma \vdash e \Rightarrow \tau}{\Sigma;\Gamma \vdash \Lambda\alpha.\,e \Rightarrow \forall\alpha.\,\tau}$$

$$\text{STAP}\quad \frac{\Sigma \vdash \tau_1 \qquad \Sigma;\Gamma \vdash e \Rightarrow \tau_2 \qquad \tau_2 \blacktriangleright_{\forall} \forall\alpha.\,\tau_3}{\Sigma;\Gamma \vdash e\,[\tau_1] \Rightarrow [\tau_1/\alpha]\tau_3}$$

$\boxed{\Sigma;\Gamma \vdash e \Leftarrow \tau}$ Expression e analyzes against type τ in context $\Sigma;\Gamma$

$$\text{ATLAM}\quad \frac{\tau_1 \blacktriangleright_{\forall} \forall\alpha.\,\tau_2 \qquad \Sigma,\alpha;\Gamma \vdash e \Leftarrow \tau_2}{\Sigma;\Gamma \vdash \Lambda\alpha.\,e \Leftarrow \tau_1}$$

Remark: STLam doubles up on introduction forms, but improves power of system.

The System

- Add elaboration and type assignment rules.
  - Nothing novel, omitted for brevity; see paper for details.
- Add instruction transitions using *type substitution*.

$\boxed{d \longrightarrow d'}$ d takes an instruction transition to d'

$$\text{ITTLam}\quad \frac{}{(\Lambda\alpha.\, d)\,[\tau] \longrightarrow [\tau/\alpha]d}$$

$$\text{ITTApCast}\quad \frac{}{d\langle\forall\alpha_1.\,\tau_1 \Rightarrow \forall\alpha_2.\,\tau_2\rangle\,[\tau] \longrightarrow d\,[\tau]\langle[\tau/\alpha_1]\tau_1 \Rightarrow [\tau/\alpha_2]\tau_2\rangle}$$

  - Previously avoided due to violating parametricity.

Properties

- Show that we are a conservative extension of the Hazelnut Live System
    - Any term typable by the bidirectional typed system has an elaboration.
    - Any elaborated term comes from a term typable by the bidirectional system.
    - The elaboration is unique.
    - The elaborated term gets assigned a type consistent with the original type.
    - Type assignment is unique.
- Anything true of the cast calculus can be lifted to the gradually typed calculus.

Properties

- Type safety: preservation & progress
- Complete type safety: If the program has no holes, execution ends in a value.
  - Generalizes System F.

- Mechanized in Agda
  - (modulo another property that isn't fully developed yet -- fill and resume)

Parametricity?

- By using substitution typing, we knowingly violate parametricity:
  - `1 <Int => ? => X>`
    - Will succeed if Int is substituted for X.
    - Will cast fail if (e.g.) Bool is substituted for X.
  - Previous authors' systems force failure in all instantiations.
- We are more permissible in what programs may be evaluated.
- Conjecture: weakening of parametricity.
  - Parametric behavior between terminating programs.
  - No guarantees for erroring executions.

Implementation

- Added to the Hazel live programming environment.
- Live demo available (guided demo now!).
- https://hazel.org/build/poly-adt-after2/
    - Currently in a PR, hopefully merge to dev soon

Demo

Implicit Polymorphism

Explicit

- Transparent
- Consistent
- Controllable

Implicit

- Compact
- Flexible

Implicit Polymorphism

Compromise: Automatic Insertions

- Compact (folding)
- Flexible (recomputed)
- Transparent
- Consistent
- Controllable

Implicit Polymorphism

*mockups

```
let map : forall X -> forall Y -> (X -> Y) -> [X] -> [Y] =  in
map (string_of_int)([1,2,3,4,5])
```

Γ EXP ? Variable reference : forall X -> forall Y -> (X -> Y) -> [X] -> [Y] with X as Int , Y as String

```
let map : forall X -> forall Y -> (X -> Y) -> [X] -> [Y] =  in
map @<Int>@<String> (string_of_int)([1,2,3,4,5])
```

Automatically inserted type application

Implicit Polymorphism

*mockups

```
let map : forall X -> forall Y -> (X -> Y) -> [X] -> [Y] = ○ in
map(string_of_int)([true, false])
```

```
Γ EXP ? Variable reference : forall X -> forall Y -> (X -> Y) -> [X] -> [Y] with X unsolved , Y as String
```

```
let map : forall X -> forall Y -> (X -> Y) -> [X] -> [Y] = ○ in
map @<!>@<String> (string_of_int)([true, false])
```

```
Γ TYP ? conflicting constraints Bool Int
```

Implicit Polymorphism

Mark Insertion

Total Type Error Localization and Recovery with Holes

ERIC ZHAO, University of Michigan, USA
RAEF MAROOF, University of Michigan, USA
ANAND DUKKIPATI, University of Michigan, USA
ANDREW BLINN, University of Michigan, USA
ZHIYI PAN, University of Michigan, USA
CYRUS OMAR, University of Michigan, USA

POPL ‘24

Implicit Polymorphism

Judgement form:

$$\Gamma \vdash e \hookrightarrow \check{e} \Rightarrow \tau$$

$$\Gamma \vdash e \hookrightarrow \check{e} \Leftarrow \tau$$

Mark Insertion

Example:

$$\frac{\Gamma \vdash e_1 \hookrightarrow \check{e}_1 \Rightarrow \tau \qquad \tau \blacktriangleright_{\not\to} \qquad \Gamma \vdash e_2 \hookrightarrow \check{e}_2 \Leftarrow ?}{\Gamma \vdash e_1\ e_2 \hookrightarrow (\!|\check{e}_1|\!)^{\Rightarrow}_{\blacktriangleright_{\not\to}}\ \check{e}_2 \Rightarrow ?}$$

Implicit Polymorphism

Type Application Insertion

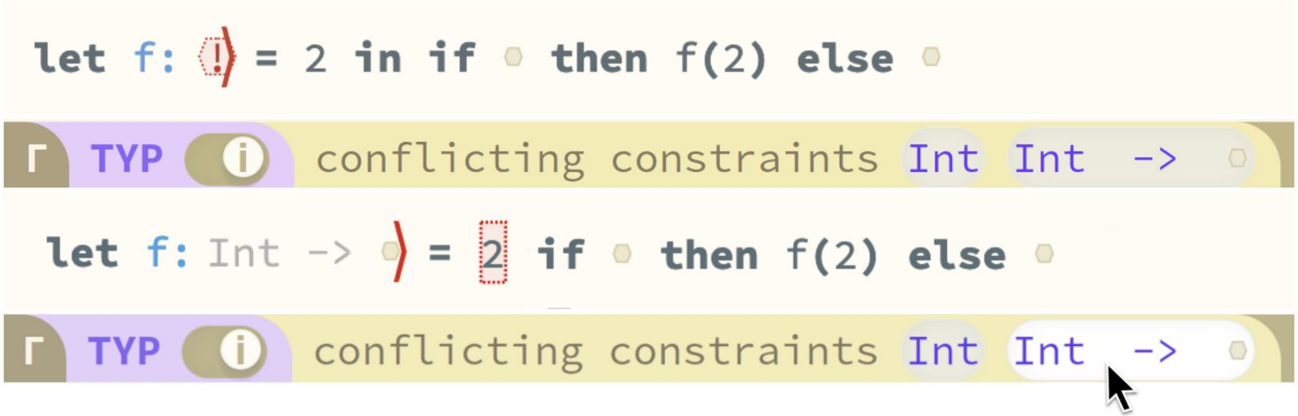
$$\frac{\Sigma;\Gamma \vdash e_1 \looparrowright \check{e}_1 \Rightarrow \tau \qquad \tau \blacktriangleright_{\not\to} \qquad \forall^{\square}(\tau) \blacktriangleright_{\to} \tau_3 \to \tau_4 \qquad \Sigma;\Gamma \vdash e_1\,[?]\,e_2 \looparrowright \check{e}_3 \Rightarrow \tau_2}{\Sigma;\Gamma \vdash e_1\,e_2 \looparrowright \check{e}_3 \Rightarrow \tau_2}$$

$$\frac{\Sigma;\Gamma \vdash e_1 \looparrowright \check{e}_1 \Rightarrow \tau \qquad \forall^{\square}(\tau) \blacktriangleright_{\not\to} \qquad \Sigma;\Gamma \vdash e_2 \looparrowright \check{e}_2 \Leftarrow ?}{\Sigma;\Gamma \vdash e_1\,e_2 \looparrowright (\!|\check{e}_1|\!)^{\Rightarrow}_{\blacktriangleright_{\not\to}}\ \check{e}_2 \Rightarrow ?}$$

Implicit Polymorphism

Type Hole Inference

POPL ‘24:



Implicit Polymorphism

Type Hole Inference

Higher order unification: undecidable

→ Eager substitution

```
let map : forall X -> forall Y -> (X -> Y) -> [X] -> [Y] = ⬡ in
map @<Int>@<String> (string_of_int)([1,2,3,4,5])
```

*mockup

🔍 Automatically inserted type application

Questions?