Total Type Error Localization and Recovery with Holes

Eric Zhao, Raef Maroof, Anand Dukkipati, Andrew Blinn, Zhiyi Pan, Cyrus Omar

Future of Programming Lab
University of Michigan

Reality check

Reality check

Most of the code we write is
ill-typed

Localization and recovery

As programmers, we want tooling that can

Localization and recovery

As programmers, we want tooling that can

- **localize type errors**

Localization and recovery

As programmers, we want tooling that can

- **localize type errors**: describe *what* and *where* they are

Localization and recovery

As programmers, we want tooling that can

- **localize type errors**: describe *what* and *where* they are
- **recover**: even when an error is encountered, continue to provision downstream services and find other errors

Localization and recovery

As programmers, we want tooling that can

- **localize type errors**: describe *what* and *where* they are
- **recover**: even when an error is encountered, continue to provision downstream services and find other errors

i.e., code completion, type hints, and other downstream services shouldn't suddenly all fail because of one error!

Observations in practice

$$\text{let } f = \lambda b : \text{bool}.\ \cdots \text{ in}$$

Observations in practice

$$\text{let } f = \lambda b : \text{bool}. \cdots \text{ in let } x = \text{if } \mathsf{true} \text{ then } 5 \text{ else } \mathsf{false} \text{ in}$$

Observations in practice

$$\text{let } f = \lambda b : \text{bool}.\ \cdots \text{ in let } x = \text{if true then } 5 \text{ else false in } f\ x$$

OCaml: the first branch is right

$$\text{let } f = \lambda b : \text{bool}.\ \cdots \text{ in let } x = \text{if true then } 5 \text{ else } \boxed{\text{false}} \text{ in } f\ x$$

```
1 | let x = if true then 5 else false in f x
                                ^^^^^
Error: This expression has type bool but an expression
       was expected of type int
```

OCaml: the first branch is right

$$\text{let } f = \lambda b : \text{bool}.\ \cdots \text{ in let } x = \text{if true then } 5 \text{ else } \boxed{\text{false}} \text{ in } f\ \boxed{x}$$

```
1 | let x = if true then 5 else false in f x
                                ^^^^^
```

```
Error: This expression has type bool but an expression
        was expected of type int
```

```
1 | let x = if true then 5 else false in f x
                                           ^
```

```
Error: This expression has type int but an expression
        was expected of type bool
```

Haskell: 5 is the problem

$$\text{let } f = \lambda b : \text{bool}. \cdots \text{ in let } x = \text{if } \text{true} \text{ then } \boxed{5} \text{ else } \text{false} \text{ in } f\ x$$

```
Main.hs:4:26: error: [GHC-39999]
    • No instance for ‘Num Bool’ arising from the literal ‘5’
  |
4 | _ = let x = if True then 5 else False in f x
  |                          ^
```

Rust: maybe 5 is also the problem?

$$\text{let } f = \lambda b : \text{bool}. \ \cdots \ \text{in let } x = \text{if true then } 5 \text{ else false in } f\ x$$

```
error[E0308]: ‘if‘ and ‘else‘ have incompatible types
 -> src/main.rs:2:30
  |
6 | let x = if true  5  else  false ; f(x);
  |                  -        ^^^^^ expected integer, found ‘bool‘
  |                  |
  |                  expected because of this
```

Hazel: the whole thing's all messed up!

$$\text{let } f = \lambda b : \text{bool}.\ \cdots \text{ in let } x = \text{if true then } 5 \text{ else false in } f\ x$$

```
let f = fun b : Bool -> ⬡ in
let x = if true then 5 else false in f(x)
```

Γ EXP ? If expression Branches have inconsistent types: Int , Bool

Observations in practice

Today's tooling is error-resilient to a certain degree

Observations in practice

Today’s tooling is error-resilient to a certain degree, but

Observations in practice

Today’s tooling is error-resilient to a certain degree, but

- localization can be varied

Observations in practice

Today's tooling is error-resilient to a certain degree, but

- localization can be varied, often guessing about *user intent*

Observations in practice

Today's tooling is error-resilient to a certain degree, but

- localization can be varied, often guessing about *user intent*
- recovery necessitates reasoning without complete knowledge about types

Observations in practice

Today's tooling is error-resilient to a certain degree, but

- localization can be varied, often guessing about *user intent*
- recovery necessitates reasoning without complete knowledge about types
- decisions can influence other downstream decisions

A definitional gap problem

Most of the code we write is *ill-typed*

A definitional gap problem

Most of the code we write is *ill-typed*, but conventional language definitions *only specify the meaning of well-typed programs.*

A definitional gap problem

Most of the code we write is *ill-typed*, but conventional language definitions *only specify the meaning of well-typed programs.*

$$\Gamma \vdash e : \tau$$

A definitional gap problem

Most of the code we write is *ill-typed*, but conventional language definitions *only specify the meaning of well-typed programs.*

$$\Gamma \vdash e : \tau$$

$$\Downarrow$$

If a type error appears **anywhere**,
the program is meaningless **everywhere**.

The goal

We'd like a way to formally specify type checkers that are capable of localizing and recovering from errors

The goal

We'd like a way to formally specify type checkers that are capable of localizing and recovering from errors

> **Totality**
>
> These semantics should give meaning to *all well-typed and ill-typed programs.*

This paper is about...

This paper is about...

Uniting *local* and *global* type inference for principled total type error localization and recovery:

This paper is about...

Uniting *local* and *global* type inference for principled total type error localization and recovery:

- the **marked lambda calculus**

This paper is about...

Uniting *local* and *global* type inference for principled total type error localization and recovery:

- the **marked lambda calculus**: a judgmental framework for bidirectional type error localization and recovery

This paper is about...

Uniting *local* and *global* type inference for principled total type error localization and recovery:

- the **marked lambda calculus**: a judgmental framework for bidirectional type error localization and recovery
- **type hole inference**

This paper is about…

Uniting *local* and *global* type inference for principled total type error localization and recovery:

- the **marked lambda calculus**: a judgmental framework for bidirectional type error localization and recovery

- **type hole inference**: a global, constraint-based system that is neutral in error localization and recovery

Marked lambda calculus: a tutorial

Start: a small gradually typed lambda calculus*

$$\begin{array}{rcl} \tau & ::= & ? \mid \mathsf{num} \mid \mathsf{bool} \mid \tau \rightarrow \tau \\ e & ::= & x \mid \lambda x : \tau .\ e \mid e\ e \mid n \mid \mathsf{true} \mid \mathsf{false} \mid \mathsf{if}\ e\ \mathsf{then}\ e\ \mathsf{else}\ e \end{array}$$

Marked lambda calculus: a tutorial

Start: a small gradually typed lambda calculus*

$$
\begin{aligned}
\tau &::= ? \mid \mathsf{num} \mid \mathsf{bool} \mid \tau \to \tau \\
e &::= x \mid \lambda x : \tau .\, e \mid e\ e \mid n \mid \mathsf{true} \mid \mathsf{false} \mid \mathsf{if}\ e\ \mathsf{then}\ e\ \mathsf{else}\ e
\end{aligned}
$$

with a standard bidirectional type system

$\Gamma \vdash e \Rightarrow \tau$ (e synthesizes type τ)

$\Gamma \vdash e \Leftarrow \tau$ (e analyzes against type τ)

Marked lambda calculus: a tutorial

Start: a small gradually typed lambda calculus*

$$
\begin{array}{rcl}
\tau & ::= & ? \mid \text{num} \mid \text{bool} \mid \tau \rightarrow \tau \\
e & ::= & x \mid \lambda x : \tau .\, e \mid e\ e \mid n \mid \text{true} \mid \text{false} \mid \text{if } e \text{ then } e \text{ else } e
\end{array}
$$

with a standard bidirectional type system

$$
\begin{array}{ll}
\Gamma \vdash e \Rightarrow \tau & (e \text{ synthesizes type } \tau) \\
\Gamma \vdash e \Leftarrow \tau & (e \text{ analyzes against type } \tau)
\end{array}
$$

*We need only the *static semantics* for ill-typed programs!

Typing variable occurrences

Synthesizing the type of a variable is standard:

$$\frac{}{\Gamma \vdash x \Rightarrow \tau}\ \textsc{SVar}$$

Typing variable occurrences

Synthesizing the type of a variable is standard:

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x \Rightarrow \tau}\;\textsc{SVar}$$

Typing variable occurrences

Synthesizing the type of a variable is standard:

$$\textsc{SVar}\ \frac{\boxed{x : \tau \in \Gamma}}{\Gamma \vdash x \Rightarrow \tau}$$

But what if $x \notin \mathrm{dom}(\Gamma)$?

Typing variable occurrences

How do we handle this failure case?

```
let rec syn ctx e =
  match e with
    Var x ->
      match Ctx.lookup ctx x with
        Some ty -> ty
        None    -> ???
    ...
```

Typing variable occurrences

How do we handle this failure case?

```
let rec syn ctx e =
  match e with
    Var x ->
      match Ctx.lookup ctx x with
        Some ty -> ty
        None    -> failwith (x ++ " is unbound")
    ...
```

Typing variable occurrences

How do we handle this failure case?

```
let rec syn ctx e =
  match e with
    Var x ->
      match Ctx.lookup ctx x with
        Some ty -> Ok(ty)
        None    -> Error(UnboundError( … ))
    ...
```

Typing variable occurrences

We've *localized* the error: "this occurrence of x is unbound!"

Typing variable occurrences

We've *localized* the error: "this occurrence of x is unbound!"

How can we *recover*?

Typing variable occurrences

We've *localized* the error: "this occurrence of x is unbound!"

How can we *recover*?

Solution. $?$

Typing variable occurrences

We've *localized* the error: "this occurrence of x is unbound!"

How can we *recover*?

Solution. $?$ $\leftarrow$ unknown type

From type checking to marking

Idea. Augment the type checking process with *marking*!

From type checking to marking

Idea. Augment the type checking process with *marking*!

- localize and report the error as a *mark*

From type checking to marking

Idea. Augment the type checking process with *marking*!

- localize and report the error as a *mark*
  - compiler messages

From type checking to marking

Idea. Augment the type checking process with *marking*!

- localize and report the error as a *mark*
    - compiler messages
    - editor decorations

From type checking to marking

Idea. Augment the type checking process with *marking*!

- localize and report the error as a *mark*
    - compiler messages
    - editor decorations

- use the unknown type to encapsulate missing type information

$$\frac{}{? \sim \tau} \qquad \frac{}{\tau \sim ?} \qquad \frac{}{\tau \sim \tau} \qquad \frac{\tau_1 \sim \tau_1' \qquad \tau_2 \sim \tau_2'}{\tau_1 \to \tau_2 \sim \tau_1' \to \tau_2'}$$

A two-layer calculus

Supplement the *unmarked* syntax

$$
\begin{aligned}
\tau &::= \; ? \mid \mathsf{num} \mid \mathsf{bool} \mid \tau \rightarrow \tau \\
e &::= \; x \mid \lambda x : \tau.\, e \mid e\ e \mid n \mid \mathsf{true} \mid \mathsf{false} \mid \mathsf{if}\ e\ \mathsf{then}\ e\ \mathsf{else}\ e
\end{aligned}
$$

A two-layer calculus

Supplement the *unmarked* syntax

$$\begin{array}{rcl} \tau & ::= & ? \mid \text{num} \mid \text{bool} \mid \tau \to \tau \\ e & ::= & x \mid \lambda x : \tau .\, e \mid e\ e \mid n \mid \text{true} \mid \text{false} \mid \text{if } e \text{ then } e \text{ else } e \end{array}$$

into a *marked* one that contains *error marks*:

$$\check{e} \ ::= \ x \mid \lambda x : \tau .\, \check{e} \mid \check{e}\ \check{e} \mid n \mid \text{true} \mid \text{false} \mid \text{if } \check{e} \text{ then } \check{e} \text{ else } \check{e} \mid (\!|x|\!)_{\square}$$

$(\!|x|\!)_{\square}$ is a *marked term* denoting a free occurrence of x

A two-layer calculus

Extend the original typing judgments

$$\Gamma \vdash e \Rightarrow \tau \quad (e \text{ synthesizes type } \tau)$$
$$\Gamma \vdash e \Leftarrow \tau \quad (e \text{ analyzes against type } \tau)$$

A two-layer calculus

Extend the original typing judgments

$$\Gamma \vdash e \Rightarrow \tau \quad (e \text{ synthesizes type } \tau)$$
$$\Gamma \vdash e \Leftarrow \tau \quad (e \text{ analyzes against type } \tau)$$

into the bidirectional **marking judgments**:

$$\Gamma \vdash e \looparrowright \check{e} \Rightarrow \tau \quad (e \text{ is marked into } \check{e} \text{ and synthesizes type } \tau)$$
$$\Gamma \vdash e \looparrowright \check{e} \Leftarrow \tau \quad (e \text{ is marked into } \check{e} \text{ and analyzes against type } \tau)$$

A two-layer calculus

Extend the original typing judgments

$$\Gamma \vdash e \Rightarrow \tau \quad (e \text{ synthesizes type } \tau)$$
$$\Gamma \vdash e \Leftarrow \tau \quad (e \text{ analyzes against type } \tau)$$

into the bidirectional **marking judgments**:

$$\Gamma \vdash e \hookrightarrow \check{e} \Rightarrow \tau \quad (e \text{ is marked into } \check{e} \text{ and synthesizes type } \tau)$$
$$\Gamma \vdash e \hookrightarrow \check{e} \Leftarrow \tau \quad (e \text{ is marked into } \check{e} \text{ and analyzes against type } \tau)$$

Type-based semantic services use marked terms!

$$\Gamma \vdash_{M} \check{e} \Rightarrow \tau \quad (\check{e} \text{ synthesizes type } \tau)$$
$$\Gamma \vdash_{M} \check{e} \Leftarrow \tau \quad (\check{e} \text{ analyzes against type } \tau)$$

Marking free variables

One typing rule for variables

$$\dfrac{x : \tau \in \Gamma}{\Gamma \vdash x \Rightarrow \tau}\ \textsc{SVar}$$

becomes two synthetic marking rules:

Marking free variables

One typing rule for variables

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x \Rightarrow \tau}\ \textsc{SVar}$$

becomes two synthetic marking rules:

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x \hookrightarrow x \Rightarrow \tau}\ \textsc{MKSVar}$$

Marking free variables

One typing rule for variables

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x \Rightarrow \tau}\ \textsc{SVar}$$

becomes two synthetic marking rules:

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x \hookrightarrow x \Rightarrow \tau}\ \textsc{MKSVar} \qquad \frac{x \notin \mathrm{dom}(\Gamma)}{\Gamma \vdash x \hookrightarrow \quad \Rightarrow}\ \textsc{MKSFree}$$

Marking free variables

One typing rule for variables

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x \Rightarrow \tau}\ \textsc{SVar}$$

becomes two synthetic marking rules:

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x \looparrowright x \Rightarrow \tau}\ \textsc{MKSVar} \qquad \frac{x \notin \mathrm{dom}(\Gamma)}{\Gamma \vdash x \looparrowright (\!|x|\!)_{\square} \Rightarrow}\ \textsc{MKSFree}$$

Marking free variables

One typing rule for variables

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x \Rightarrow \tau}\ \textsc{SVar}$$

becomes two synthetic marking rules:

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x \looparrowright x \Rightarrow \tau}\ \textsc{MKSVar}
\qquad
\frac{x \notin \mathrm{dom}(\Gamma)}{\Gamma \vdash x \looparrowright (\!|x|\!)_{\square} \Rightarrow\ ?}\ \textsc{MKSFree}$$

Marking local inconsistencies

The standard subsumption principle:

$$\frac{\Gamma \vdash e \Rightarrow \tau' \qquad \tau \sim \tau'}{\Gamma \vdash e \Leftarrow \tau}\ \textsc{UASubsume}$$

Marking local inconsistencies

The standard subsumption principle:

$$\frac{\Gamma \vdash e \Rightarrow \tau' \qquad \tau \sim \tau'}{\Gamma \vdash e \Leftarrow \tau}\ \textsc{UASubsume}$$

becomes the analytic marking rule:

$$\frac{\Gamma \vdash e \looparrowright \check{e} \Rightarrow \tau' \qquad \tau \sim \tau'}{\Gamma \vdash e \looparrowright \check{e} \Leftarrow \tau}\ \textsc{MKASubsume}$$

Marking local inconsistencies

What if $\tau \nsim \tau'$, *e.g.*, in $5 + \mathsf{true}$?

Marking local inconsistencies

What if $\tau \nsim \tau'$, *e.g.*, in $5 + \mathsf{true}$?

$$\check{e} ::= \cdots \mid (\!|x|\!)_{\square} \mid (\!|\check{e}|\!)_{\nsim}$$

Marking local inconsistencies

What if $\tau \nsim \tau'$, *e.g.*, in $5 + \mathsf{true}$?

$$\check{e} ::= \cdots \mid (\!|x|\!)_{\square} \mid (\!|\check{e}|\!)_{\nsim}$$

$$\frac{\Gamma \vdash e \looparrowright \check{e} \Rightarrow \tau' \qquad \tau \sim \tau'}{\Gamma \vdash e \looparrowright \check{e} \Leftarrow \tau}\ \textsc{MKASubsume}$$

$$\frac{\Gamma \vdash e \looparrowright \check{e} \Rightarrow \tau' \qquad \tau \nsim \tau'}{\Gamma \vdash e \looparrowright (\!|\check{e}|\!)_{\nsim} \Leftarrow \tau}\ \textsc{MKAInconsistentTypes}$$

Marking local inconsistencies

What if $\tau \nsim \tau'$, *e.g.*, in $5 + (\!|\mathsf{true}|\!)_{\nsim}$?

$$\check{e} ::= \cdots \mid (\!|x|\!)_{\square} \mid (\!|\check{e}|\!)_{\nsim}$$

$$\frac{\Gamma \vdash e \hookrightarrow \check{e} \Rightarrow \tau' \qquad \tau \sim \tau'}{\Gamma \vdash e \hookrightarrow \check{e} \Leftarrow \tau}\ \textsc{MKASubsume} \qquad \frac{\Gamma \vdash e \hookrightarrow \check{e} \Rightarrow \tau' \qquad \tau \nsim \tau'}{\Gamma \vdash e \hookrightarrow (\!|\check{e}|\!)_{\nsim} \Leftarrow \tau}\ \textsc{MKAInconsistentTypes}$$

… and the rest

$$
\begin{array}{rcll}
e & ::= & \cdots & \\
& | & (\!|\check{e}_1|\!)^{\Rightarrow}_{\blacktriangleright_{\not\to}}\ \check{e}_2 & \check{e}_1 \text{ synthesizes non-matched arrow type} \\
& | & (\!|\mathbf{if}\ \check{e}_1\ \mathbf{then}\ \check{e}_2\ \mathbf{else}\ \check{e}_3|\!)_{\not\sqcup} & \text{branches synthesize inconsistent types} \\
& | & (\!|\lambda x : \tau.\ \check{e}|\!)^{\Leftarrow}_{\blacktriangleright_{\not\to}} & \text{analysis against non-matched arrow type} \\
& | & (\!|\lambda x : \tau.\ \check{e}|\!)_{:} & \text{ascription inconsistent with domain}
\end{array}
$$

A total marking

> **Totality**
>
> These semantics should give meaning to *all well-typed and ill-typed programs.*

A total marking

Totality

These semantics should give meaning to *all well-typed and ill-typed programs.*

$$\Downarrow$$

Theorem 2.1. Totality

For all Γ and e, $\exists \check{e}, \tau$ s.t. $\Gamma \vdash e \looparrowright \check{e} \Rightarrow \tau$.
For all Γ, e, and τ, $\exists \check{e}$ s.t. $\Gamma \vdash e \looparrowright \check{e} \Leftarrow \tau$.
(Every unmarked term can be marked under any context!)

A sensible marking

Theorem 2.2. Well-Formedness

If $\Gamma \vdash e \looparrowright \check{e} \Rightarrow \tau$, then $\Gamma \vdash_{M} \check{e} \Rightarrow \tau$ and $\check{e}^{\square} = e$.
If $\Gamma \vdash e \looparrowright \check{e} \Leftarrow \tau$, then $\Gamma \vdash_{M} \check{e} \Leftarrow \tau$ and $\check{e}^{\square} = e$.
(Marking only adds marks, *i.e.*, marking then erasing is identity!)

A sensible marking

> **Theorem 2.2. Well-Formedness**
>
> If $\Gamma \vdash e \looparrowright \check{e} \Rightarrow \tau$, then $\Gamma \vdash_{M} \check{e} \Rightarrow \tau$ and $\check{e}^{\square} = e$.
> If $\Gamma \vdash e \looparrowright \check{e} \Leftarrow \tau$, then $\Gamma \vdash_{M} \check{e} \Leftarrow \tau$ and $\check{e}^{\square} = e$.
> (Marking only adds marks, *i.e.*, marking then erasing is identity!)

A sensible marking

Theorem 2.2. Well-Formedness

If $\Gamma \vdash e \looparrowright \check{e} \Rightarrow \tau$, then $\Gamma \vdash_{M} \check{e} \Rightarrow \tau$ and $\check{e}^{\square} = e$.
If $\Gamma \vdash e \looparrowright \check{e} \Leftarrow \tau$, then $\Gamma \vdash_{M} \check{e} \Leftarrow \tau$ and $\check{e}^{\square} = e$.
(Marking only adds marks, *i.e.*, marking then erasing is identity!)

Theorem 2.3(1). Well-Typed Terms

If $\Gamma \vdash_{U} e \Rightarrow \tau$, then $\exists \check{e}$ s.t. $\Gamma \vdash e \looparrowright \check{e} \Rightarrow \tau$ and $\check{e}$ markless.
If $\Gamma \vdash_{U} e \Leftarrow \tau$, then $\exists \check{e}$ s.t. $\Gamma \vdash e \looparrowright \check{e} \Leftarrow \tau$ and $\check{e}$ markless.
(Marking well-typed terms introduces no marks!)

A sensible marking

Theorem 2.2. Well-Formedness

If $\Gamma \vdash e \looparrowright \check{e} \Rightarrow \tau$, then $\Gamma \vdash_{M} \check{e} \Rightarrow \tau$ and $\check{e}^{\square} = e$.
If $\Gamma \vdash e \looparrowright \check{e} \Leftarrow \tau$, then $\Gamma \vdash_{M} \check{e} \Leftarrow \tau$ and $\check{e}^{\square} = e$.
(Marking only adds marks, *i.e.*, marking then erasing is identity!)

Theorem 2.3(1). Well-Typed Terms

If $\Gamma \vdash_{U} e \Rightarrow \tau$, then $\exists \check{e}$ s.t. $\Gamma \vdash e \looparrowright \check{e} \Rightarrow \tau$ and $\check{e}$ markless.
If $\Gamma \vdash_{U} e \Leftarrow \tau$, then $\exists \check{e}$ s.t. $\Gamma \vdash e \looparrowright \check{e} \Leftarrow \tau$ and $\check{e}$ markless.
(Marking well-typed terms introduces no marks!)

A sensible marking

e

(*marking*)

$$\Gamma \vdash e \looparrowright \check{e} \Rightarrow \tau$$
$$\Gamma \vdash e \looparrowright \check{e} \Leftarrow \tau$$

(*erasure*) $\_^{\square}$

$$\Gamma \vdash_{M} \check{e} \Rightarrow \tau$$
$$\Gamma \vdash_{M} \check{e} \Leftarrow \tau \dashrightarrow \text{(downstream services)}$$

(*marked*)

A sensible marking

e

(marking)

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

(erasure) $\_^{\square}$

$$\Gamma \vdash_{M} \check{e} \Rightarrow \tau$$
$$\Gamma \vdash_{M} \check{e} \Leftarrow \tau \dashrightarrow \text{(downstream services)}$$

(marked)

Metatheory completely mechanized in Agda [hazelgrove/error-localization-agda]

The Recipe

- Begin with a bidirectional gradually* typed language
  - *Only need the static parts for marking ill-typed programs!

The Recipe

- Begin with a bidirectional gradually* typed language
  - *Only need the static parts for marking ill-typed programs!
- Derive marking rules from each typing rule
  - Consider the “success” case
  - Consider the “failure” cases, introducing error marks

The Recipe

- Begin with a bidirectional gradually* typed language
  - *Only need the static parts for marking ill-typed programs!
- Derive marking rules from each typing rule
  - Consider the “success” case
  - Consider the “failure” cases, introducing error marks
- *Not* prescriptive *w.r.t.* localization strategy
  - We formalize three possible localization strategies for if-then-else with inconsistent branches

Marking global inconsistencies?

Consider this program:

$$\lambda f : ?\,.\ f\ (f + 1)$$

$f : ?$, so the bidirectional system operates gradually,

Marking global inconsistencies?

Consider this program:

$$\lambda f : ?\,.\ f\ (f + 1)$$

$f : ?$, so the bidirectional system operates gradually, but f is

1. applied as a function (a function?)

Marking global inconsistencies?

Consider this program:

$$\lambda f : ?\,.\; f\ (f + 1)$$

$f\ :\ ?$, so the bidirectional system operates gradually, but f is

1. applied as a function (a function?)
2. an operand of $+$ (a number?)

Marking global inconsistencies?

Consider this program:

$$\lambda f : ?\,.\; f\;(f + 1)$$

$f : ?$, so the bidirectional system operates gradually, but f is

1. applied as a function (a function?)
2. an operand of $+$ (a number?)

Global, constraint-based type checking would have caught this!

Layers upon layers

Get the best of both worlds by layering **constraint-based inference** atop the marked lambda calculus

Layers upon layers

Get the best of both worlds by layering **constraint-based inference** atop the marked lambda calculus

- The marked lambda calculus localizes and recovers predictably from *local inconsistencies*

Layers upon layers

Get the best of both worlds by layering **constraint-based inference** atop the marked lambda calculus

- The marked lambda calculus localizes and recovers predictably from *local inconsistencies*
- **Type hole inference** solves and marks *global inconsistencies*

Layers upon layers

Get the best of both worlds by layering **constraint-based inference** atop the marked lambda calculus

- The marked lambda calculus localizes and recovers predictably from *local inconsistencies*

- **Type hole inference** solves and marks *global inconsistencies*
  - Downstream service to supplement the marked lambda calculus

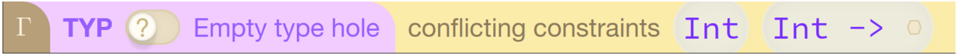
Type hole inference in Hazel

```
fun f : ! -> f (f + 1)
```

Γ TYP ? Empty type hole conflicting constraints Int Int ->

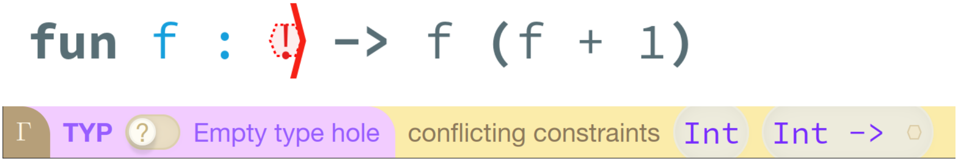
Type hole inference in Hazel

```
fun f : ! -> f (f + 1)
```



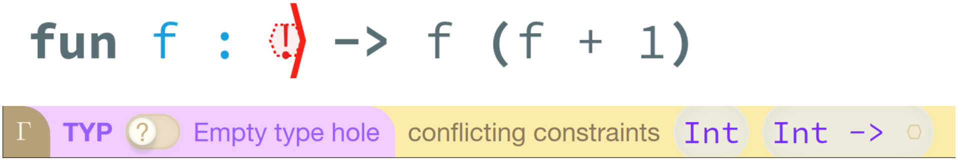
1. Gather constraints, treat occurrences of ? as unification variables

Type hole inference in Hazel



1. Gather constraints, treat occurrences of ? as unification variables
2. When solvable, proceed as normal to find substitution

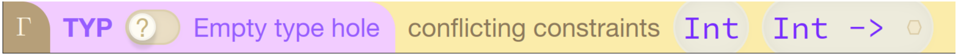
Type hole inference in Hazel



1. Gather constraints, treat occurrences of ? as unification variables
2. When solvable, proceed as normal to find substitution
3. When unsolvable, maintain partial solutions: possible type fillings, which are offered to the user for selection

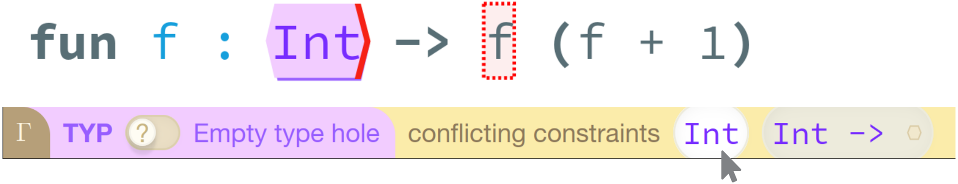
Type hole inference in Hazel

```
fun f : ! -> f (f + 1)
```



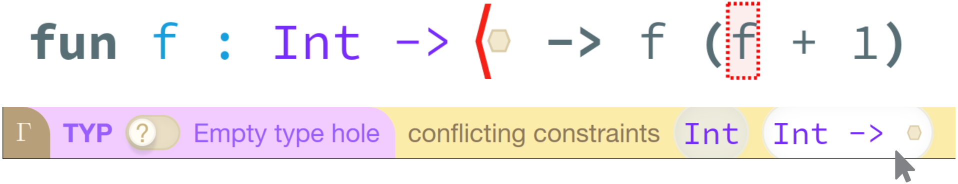
1. Gather constraints, treat occurrences of ? as unification variables
2. When solvable, proceed as normal to find substitution
3. When unsolvable, maintain partial solutions: possible type fillings, which are offered to the user for selection
4. Control returned to the bidirectional system upon selection

Type hole inference in Hazel



1. Gather constraints, treat occurrences of ? as unification variables
2. When solvable, proceed as normal to find substitution
3. When unsolvable, maintain partial solutions: possible type fillings, which are offered to the user for selection
4. Control returned to the bidirectional system upon selection

Type hole inference in Hazel



1. Gather constraints, treat occurrences of ? as unification variables
2. When solvable, proceed as normal to find substitution
3. When unsolvable, maintain partial solutions: possible type fillings, which are offered to the user for selection
4. Control returned to the bidirectional system upon selection

Type hole inference in Hazel

This approach is *neutral*

Type hole inference in Hazel

This approach is *neutral*

- Localize errors to the originating type hole or error mark, instead of guessing about user intent

Type hole inference in Hazel

This approach is *neutral*

- Localize errors to the originating type hole or error mark, instead of guessing about user intent

- All potential type hole fillings are provided to the user for selection

Type hole inference in Hazel

This approach is *neutral*

- Localize errors to the originating type hole or error mark, instead of guessing about user intent
- All potential type hole fillings are provided to the user for selection
- Control returned to bidirectional system after user selects

More in the paper and artifact

- A full description of the marked lambda calculus

More in the paper and artifact

- A full description of the marked lambda calculus, and
  - mechanization in Agda [hazelgrove/error-localization-agda]

More in the paper and artifact

- A full description of the marked lambda calculus, and
  - mechanization in Agda [hazelgrove/error-localization-agda]
  - extensions to richer typing features

More in the paper and artifact

- A full description of the marked lambda calculus, and
  - mechanization in Agda [hazelgrove/error-localization-agda]
  - extensions to richer typing features
    (parametric polymorphism and destructuring let)

More in the paper and artifact

- A full description of the marked lambda calculus, and
  - mechanization in Agda [hazelgrove/error-localization-agda]
  - extensions to richer typing features
    (parametric polymorphism and destructuring let)
  - connections to structured editing

More in the paper and artifact

- A full description of the marked lambda calculus, and
  - mechanization in Agda [hazelgrove/error-localization-agda]
  - extensions to richer typing features
    (parametric polymorphism and destructuring let)
  - connections to structured editing

- A more thorough discussion of type hole inference

More in the paper and artifact

- A full description of the marked lambda calculus, and
  - mechanization in Agda [hazelgrove/error-localization-agda]
  - extensions to richer typing features
    (parametric polymorphism and destructuring let)
  - connections to structured editing
- A more thorough discussion of type hole inference, and
  - filling expression holes

More in the paper and artifact

- A full description of the marked lambda calculus, and
  - mechanization in Agda [hazelgrove/error-localization-agda]
  - extensions to richer typing features (parametric polymorphism and destructuring let)
  - connections to structured editing
- A more thorough discussion of type hole inference, and
  - filling expression holes
  - polymorphic generalization

More in the paper and artifact

- A full description of the marked lambda calculus, and
  - mechanization in Agda [hazelgrove/error-localization-agda]
  - extensions to richer typing features (parametric polymorphism and destructuring let)
  - connections to structured editing
- A more thorough discussion of type hole inference, and
  - filling expression holes
  - polymorphic generalization
- Implementations of both in Hazel [hazel.org] [hazelgrove/hazel]

More in the paper and artifact

- A full description of the marked lambda calculus, and
  - mechanization in Agda [hazelgrove/error-localization-agda]
  - extensions to richer typing features (parametric polymorphism and destructuring let)
  - connections to structured editing
- A more thorough discussion of type hole inference, and
  - filling expression holes
  - polymorphic generalization
- Implementations of both in Hazel [hazel.org] [hazelgrove/hazel]

Consider using these techniques for your next language!

Marking many ways

MKSIF

$$\frac{}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \hookrightarrow \qquad\qquad \Rightarrow}$$

$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$

Marking many ways

$$\dfrac{\Gamma \vdash e_1 \hookrightarrow \check{e}_1 \Leftarrow \mathsf{bool}}{\Gamma \vdash \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \hookrightarrow \qquad \Rightarrow}\ \textsc{MKSIf}$$

$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$

Marking many ways

MKSIf

$$\frac{\Gamma \vdash e_1 \hookrightarrow \check{e}_1 \Leftarrow \mathsf{bool} \qquad \Gamma \vdash e_2 \hookrightarrow \check{e}_2 \Rightarrow \tau_1}{\Gamma \vdash \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \hookrightarrow \qquad\qquad \Rightarrow}$$

. . . .

Marking many ways

MKSIF

$$\frac{\Gamma \vdash e_1 \looparrowright \check{e}_1 \Leftarrow \text{bool} \qquad \Gamma \vdash e_2 \looparrowright \check{e}_2 \Rightarrow \tau_1 \qquad \Gamma \vdash e_3 \looparrowright \check{e}_3 \Rightarrow \tau_2}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \looparrowright \qquad\qquad \Rightarrow}$$

Marking many ways

MKSIf

$$\frac{\Gamma \vdash e_1 \hookrightarrow \check{e}_1 \Leftarrow \text{bool} \qquad \Gamma \vdash e_2 \hookrightarrow \check{e}_2 \Rightarrow \tau_1 \qquad \Gamma \vdash e_3 \hookrightarrow \check{e}_3 \Rightarrow \tau_2 \qquad \tau_3 = \tau_1 \sqcap \tau_2}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \hookrightarrow \qquad\qquad \Rightarrow}$$

. . . .

Marking many ways

MKSIf

$$\frac{\Gamma \vdash e_1 \hookrightarrow \check{e}_1 \Leftarrow \mathsf{bool} \qquad \Gamma \vdash e_2 \hookrightarrow \check{e}_2 \Rightarrow \tau_1 \qquad \Gamma \vdash e_3 \hookrightarrow \check{e}_3 \Rightarrow \tau_2 \qquad \tau_3 = \tau_1 \sqcap \tau_2}{\Gamma \vdash \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \hookrightarrow \mathsf{if}\ \check{e}_1\ \mathsf{then}\ \check{e}_2\ \mathsf{else}\ \check{e}_3 \Rightarrow}$$

. . . .

Marking many ways

MKSIF

$$\frac{\Gamma \vdash e_1 \looparrowright \check{e}_1 \Leftarrow \mathsf{bool} \qquad \Gamma \vdash e_2 \looparrowright \check{e}_2 \Rightarrow \tau_1 \qquad \Gamma \vdash e_3 \looparrowright \check{e}_3 \Rightarrow \tau_2 \qquad \tau_3 = \tau_1 \sqcap \tau_2}{\Gamma \vdash \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \looparrowright \mathsf{if}\ \check{e}_1\ \mathsf{then}\ \check{e}_2\ \mathsf{else}\ \check{e}_3 \Rightarrow \tau_3}$$

. . . .

Marking many ways

MKSIf

$$\frac{\Gamma \vdash e_1 \hookrightarrow \check{e}_1 \Leftarrow \text{bool} \qquad \Gamma \vdash e_2 \hookrightarrow \check{e}_2 \Rightarrow \tau_1 \qquad \Gamma \vdash e_3 \hookrightarrow \check{e}_3 \Rightarrow \tau_2 \qquad \tau_3 = \tau_1 \sqcap \tau_2}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \hookrightarrow \text{if } \check{e}_1 \text{ then } \check{e}_2 \text{ else } \check{e}_3 \Rightarrow \tau_3}$$

MKSInconsistentBranches

$$\frac{\Gamma \vdash e_1 \hookrightarrow \check{e}_1 \Leftarrow \text{bool} \qquad \Gamma \vdash e_2 \hookrightarrow \check{e}_2 \Rightarrow \tau_1 \qquad \Gamma \vdash e_3 \hookrightarrow \check{e}_3 \Rightarrow \tau_2 \qquad \tau_1 \nsim \tau_2}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \hookrightarrow \qquad\qquad\qquad\qquad \Rightarrow}$$

. . . .

Marking many ways

$$\dfrac{\Gamma \vdash e_1 \looparrowright \check{e}_1 \Leftarrow \text{bool} \qquad \Gamma \vdash e_2 \looparrowright \check{e}_2 \Rightarrow \tau_1 \qquad \Gamma \vdash e_3 \looparrowright \check{e}_3 \Rightarrow \tau_2 \qquad \tau_3 = \tau_1 \sqcap \tau_2}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \looparrowright \text{if } \check{e}_1 \text{ then } \check{e}_2 \text{ else } \check{e}_3 \Rightarrow \tau_3}\ \textsc{MKSIf}$$

$$\dfrac{\Gamma \vdash e_1 \looparrowright \check{e}_1 \Leftarrow \text{bool} \qquad \Gamma \vdash e_2 \looparrowright \check{e}_2 \Rightarrow \tau_1 \qquad \Gamma \vdash e_3 \looparrowright \check{e}_3 \Rightarrow \tau_2 \qquad \tau_1 \nsim \tau_2}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \looparrowright (\!| \text{if } \check{e}_1 \text{ then } \check{e}_2 \text{ else } \check{e}_3 |\!)_{\not\sqcap} \Rightarrow}\ \textsc{MKSInconsistentBranches}$$

. . . .

Marking many ways

MKSIf

$$\frac{\Gamma \vdash e_1 \hookrightarrow \check{e}_1 \Leftarrow \text{bool} \qquad \Gamma \vdash e_2 \hookrightarrow \check{e}_2 \Rightarrow \tau_1 \qquad \Gamma \vdash e_3 \hookrightarrow \check{e}_3 \Rightarrow \tau_2 \qquad \tau_3 = \tau_1 \sqcap \tau_2}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \hookrightarrow \text{if } \check{e}_1 \text{ then } \check{e}_2 \text{ else } \check{e}_3 \Rightarrow \tau_3}$$

MKSInconsistentBranches

$$\frac{\Gamma \vdash e_1 \hookrightarrow \check{e}_1 \Leftarrow \text{bool} \qquad \Gamma \vdash e_2 \hookrightarrow \check{e}_2 \Rightarrow \tau_1 \qquad \Gamma \vdash e_3 \hookrightarrow \check{e}_3 \Rightarrow \tau_2 \qquad \tau_1 \nsim \tau_2}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \hookrightarrow (\!|\text{if } \check{e}_1 \text{ then } \check{e}_2 \text{ else } \check{e}_3|\!)_{\not\sqcap} \Rightarrow ?}$$

. . . .

Marking many ways

MKSIf

$$\frac{\Gamma \vdash e_1 \hookrightarrow \check{e}_1 \Leftarrow \text{bool} \qquad \Gamma \vdash e_2 \hookrightarrow \check{e}_2 \Rightarrow \tau_1 \qquad \Gamma \vdash e_3 \hookrightarrow \check{e}_3 \Rightarrow \tau_2 \qquad \tau_3 = \tau_1 \sqcap \tau_2}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \hookrightarrow \text{if } \check{e}_1 \text{ then } \check{e}_2 \text{ else } \check{e}_3 \Rightarrow \tau_3}$$

MKSInconsistentBranches

$$\frac{\Gamma \vdash e_1 \hookrightarrow \check{e}_1 \Leftarrow \text{bool} \qquad \Gamma \vdash e_2 \hookrightarrow \check{e}_2 \Rightarrow \tau_1 \qquad \Gamma \vdash e_3 \hookrightarrow \check{e}_3 \Rightarrow \tau_2 \qquad \tau_1 \nsim \tau_2}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \hookrightarrow (\!| \text{if } \check{e}_1 \text{ then } \check{e}_2 \text{ else } \check{e}_3 |\!)_{\not\sqcap} \Rightarrow \ ?}$$

. . . .

MKSIf’

$$\frac{}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \hookrightarrow \qquad \Rightarrow}$$

Marking many ways

MKSIf

$$\frac{\Gamma \vdash e_1 \looparrowright \check{e}_1 \Leftarrow \text{bool} \qquad \Gamma \vdash e_2 \looparrowright \check{e}_2 \Rightarrow \tau_1 \qquad \Gamma \vdash e_3 \looparrowright \check{e}_3 \Rightarrow \tau_2 \qquad \tau_3 = \tau_1 \sqcap \tau_2}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \looparrowright \text{if } \check{e}_1 \text{ then } \check{e}_2 \text{ else } \check{e}_3 \Rightarrow \tau_3}$$

MKSInconsistentBranches

$$\frac{\Gamma \vdash e_1 \looparrowright \check{e}_1 \Leftarrow \text{bool} \qquad \Gamma \vdash e_2 \looparrowright \check{e}_2 \Rightarrow \tau_1 \qquad \Gamma \vdash e_3 \looparrowright \check{e}_3 \Rightarrow \tau_2 \qquad \tau_1 \nsim \tau_2}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \looparrowright (\!|\text{if } \check{e}_1 \text{ then } \check{e}_2 \text{ else } \check{e}_3|\!)_{\not\sim} \Rightarrow ?}$$

. . . .

MKSIf'

$$\frac{\Gamma \vdash e_1 \looparrowright \check{e}_1 \Leftarrow \text{bool}}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \looparrowright \qquad\qquad\qquad\qquad \Rightarrow}$$

Marking many ways

$$\frac{\Gamma \vdash e_1 \hookrightarrow \check{e}_1 \Leftarrow \text{bool} \qquad \Gamma \vdash e_2 \hookrightarrow \check{e}_2 \Rightarrow \tau_1 \qquad \Gamma \vdash e_3 \hookrightarrow \check{e}_3 \Rightarrow \tau_2 \qquad \tau_3 = \tau_1 \sqcap \tau_2}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \hookrightarrow \text{if } \check{e}_1 \text{ then } \check{e}_2 \text{ else } \check{e}_3 \Rightarrow \tau_3}\ \textsc{MKSIf}$$

$$\frac{\Gamma \vdash e_1 \hookrightarrow \check{e}_1 \Leftarrow \text{bool} \qquad \Gamma \vdash e_2 \hookrightarrow \check{e}_2 \Rightarrow \tau_1 \qquad \Gamma \vdash e_3 \hookrightarrow \check{e}_3 \Rightarrow \tau_2 \qquad \tau_1 \nsim \tau_2}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \hookrightarrow (\!|\text{if } \check{e}_1 \text{ then } \check{e}_2 \text{ else } \check{e}_3|\!)_{\not\sqcap} \Rightarrow \,?}\ \textsc{MKSInconsistentBranches}$$

. . . .

MKSIf'

(prioritize first branch)

$$\frac{\Gamma \vdash e_1 \hookrightarrow \check{e}_1 \Leftarrow \text{bool} \qquad \Gamma \vdash e_2 \hookrightarrow \check{e}_2 \Rightarrow \tau}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \hookrightarrow \qquad\qquad \Rightarrow}$$

Marking many ways

MKSIf

$$\frac{\Gamma \vdash e_1 \hookrightarrow \check{e}_1 \Leftarrow \text{bool} \qquad \Gamma \vdash e_2 \hookrightarrow \check{e}_2 \Rightarrow \tau_1 \qquad \Gamma \vdash e_3 \hookrightarrow \check{e}_3 \Rightarrow \tau_2 \qquad \tau_3 = \tau_1 \sqcap \tau_2}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \hookrightarrow \text{if } \check{e}_1 \text{ then } \check{e}_2 \text{ else } \check{e}_3 \Rightarrow \tau_3}$$

MKSInconsistentBranches

$$\frac{\Gamma \vdash e_1 \hookrightarrow \check{e}_1 \Leftarrow \text{bool} \qquad \Gamma \vdash e_2 \hookrightarrow \check{e}_2 \Rightarrow \tau_1 \qquad \Gamma \vdash e_3 \hookrightarrow \check{e}_3 \Rightarrow \tau_2 \qquad \tau_1 \nsim \tau_2}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \hookrightarrow (\!|\text{if } \check{e}_1 \text{ then } \check{e}_2 \text{ else } \check{e}_3|\!)_{\not\sqcap} \Rightarrow \ ?}$$

. . . .

MKSIf'

(prioritize first branch) (blame second branch)

$$\frac{\Gamma \vdash e_1 \hookrightarrow \check{e}_1 \Leftarrow \text{bool} \qquad \boxed{\Gamma \vdash e_2 \hookrightarrow \check{e}_2 \Rightarrow \tau} \qquad \boxed{\Gamma \vdash e_3 \hookrightarrow \check{e}_3 \Leftarrow \tau}}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \hookrightarrow \qquad\qquad\qquad\qquad \Rightarrow}$$

Marking many ways

MKSIf

$$\frac{\Gamma \vdash e_1 \looparrowright \check{e}_1 \Leftarrow \text{bool} \qquad \Gamma \vdash e_2 \looparrowright \check{e}_2 \Rightarrow \tau_1 \qquad \Gamma \vdash e_3 \looparrowright \check{e}_3 \Rightarrow \tau_2 \qquad \tau_3 = \tau_1 \sqcap \tau_2}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \looparrowright \text{if } \check{e}_1 \text{ then } \check{e}_2 \text{ else } \check{e}_3 \Rightarrow \tau_3}$$

MKSInconsistentBranches

$$\frac{\Gamma \vdash e_1 \looparrowright \check{e}_1 \Leftarrow \text{bool} \qquad \Gamma \vdash e_2 \looparrowright \check{e}_2 \Rightarrow \tau_1 \qquad \Gamma \vdash e_3 \looparrowright \check{e}_3 \Rightarrow \tau_2 \qquad \tau_1 \nsim \tau_2}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \looparrowright (\!|\text{if } \check{e}_1 \text{ then } \check{e}_2 \text{ else } \check{e}_3|\!)_{\not\sqcap} \Rightarrow \ ?}$$

. . . .

MKSIf'

(prioritize first branch) (blame second branch)

$$\frac{\Gamma \vdash e_1 \looparrowright \check{e}_1 \Leftarrow \text{bool} \qquad \Gamma \vdash e_2 \looparrowright \check{e}_2 \Rightarrow \tau \qquad \Gamma \vdash e_3 \looparrowright \check{e}_3 \Leftarrow \tau}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \looparrowright \text{if } \check{e}_1 \text{ then } \check{e}_2 \text{ else } \check{e}_3 \Rightarrow}$$

Marking many ways

MKSIf

$$\frac{\Gamma \vdash e_1 \hookrightarrow \check{e}_1 \Leftarrow \text{bool} \qquad \Gamma \vdash e_2 \hookrightarrow \check{e}_2 \Rightarrow \tau_1 \qquad \Gamma \vdash e_3 \hookrightarrow \check{e}_3 \Rightarrow \tau_2 \qquad \tau_3 = \tau_1 \sqcap \tau_2}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \hookrightarrow \text{if } \check{e}_1 \text{ then } \check{e}_2 \text{ else } \check{e}_3 \Rightarrow \tau_3}$$

MKSInconsistentBranches

$$\frac{\Gamma \vdash e_1 \hookrightarrow \check{e}_1 \Leftarrow \text{bool} \qquad \Gamma \vdash e_2 \hookrightarrow \check{e}_2 \Rightarrow \tau_1 \qquad \Gamma \vdash e_3 \hookrightarrow \check{e}_3 \Rightarrow \tau_2 \qquad \tau_1 \nsim \tau_2}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \hookrightarrow (\!|\text{if } \check{e}_1 \text{ then } \check{e}_2 \text{ else } \check{e}_3|\!)_{\not\sqcap} \Rightarrow\ ?}$$

. . . .

MKSIf'

(prioritize first branch) (blame second branch)

$$\frac{\Gamma \vdash e_1 \hookrightarrow \check{e}_1 \Leftarrow \text{bool} \qquad \Gamma \vdash e_2 \hookrightarrow \check{e}_2 \Rightarrow \tau \qquad \Gamma \vdash e_3 \hookrightarrow \check{e}_3 \Leftarrow \tau}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \hookrightarrow \text{if } \check{e}_1 \text{ then } \check{e}_2 \text{ else } \check{e}_3 \Rightarrow \tau}$$

Provenances for types

Add type provenances p to the *marked language*:

Provenances for types

Add type provenances p to the *marked language*:

- Link occurrences of $?$ to the originating type hole or error mark (denoted by unique id u)

- Distinguish between different $?$ based on locus

Provenances for types

Add type provenances p to the *marked language*:

- Link occurrences of $?$ to the originating type hole or error mark (denoted by unique id u)

- Distinguish between different $?$ based on locus

$$p \quad ::= \quad u \mid exp(u) \mid \rightarrow_L (p) \mid \rightarrow_R (p)$$

Provenances for types

Add type provenances p to the *marked language*:

- Link occurrences of $?$ to the originating type hole or error mark (denoted by unique id u)

- Distinguish between different $?$ based on locus

$$
\begin{aligned}
p &::= u \mid exp(u) \mid \rightarrow_L (p) \mid \rightarrow_R (p) \\
\tau &::= \cdots \mid ?^p
\end{aligned}
$$

Provenances for types

Add type provenances p to the *marked language*:

- Link occurrences of $?$ to the originating type hole or error mark (denoted by unique id u)
- Distinguish between different $?$ based on locus

$$
\begin{array}{lcl}
p & ::= & u \mid exp(u) \mid \rightarrow_L (p) \mid \rightarrow_R (p) \\
\tau & ::= & \cdots \mid ?^p \\
\check{e} & ::= & \cdots \mid (\!| x |\!)^u_\square \mid (\!| \check{e} |\!)^{\Rightarrow,\ u}_{\blacktriangleright_{\not\to}} \mid (\!| \check{e} |\!)^u_{\not\sim}
\end{array}
$$

Generating constraints bidirectionally

Augment the type system (of the marked language) to generate sets C of constraints $\tau_1 \approx \tau_2$

Generating constraints bidirectionally

Augment the type system (of the marked language) to generate sets C of constraints $\tau_1 \approx \tau_2$, which force τ_1 and τ_2 to be consistent:

Generating constraints bidirectionally

Augment the type system (of the marked language) to generate sets C of constraints $\tau_1 \approx \tau_2$, which force τ_1 and τ_2 to be consistent:

$$\Gamma \vdash \check{e} \Rightarrow \tau \mid C \qquad\qquad \Gamma \vdash \check{e} \Leftarrow \tau \mid C$$

Generating constraints bidirectionally

Augment the type system (of the marked language) to generate sets C of constraints $\tau_1 \approx \tau_2$, which force τ_1 and τ_2 to be consistent:

$$\Gamma \vdash \check{e} \Rightarrow \tau \mid C \qquad\qquad \Gamma \vdash \check{e} \Leftarrow \tau \mid C$$

$$\textsc{MASubsume-C}\quad \frac{\Gamma \vdash \check{e} \Rightarrow \tau' \mid C \qquad \tau \sim \tau'}{\Gamma \vdash \check{e} \Leftarrow \tau \mid C \cup \{\tau \approx \tau'\}}$$

Generating constraints bidirectionally

Augment the type system (of the marked language) to generate sets C of constraints $\tau_1 \approx \tau_2$, which force τ_1 and τ_2 to be consistent:

$$\Gamma \vdash \check{e} \Rightarrow \tau \mid C \qquad\qquad \Gamma \vdash \check{e} \Leftarrow \tau \mid C$$

$$\frac{\Gamma \vdash \check{e} \Rightarrow \tau' \mid C \qquad \tau \sim \tau'}{\Gamma \vdash \check{e} \Leftarrow \tau \mid C \cup \{\tau \approx \tau'\}}\ \textsc{MASubsume-C}$$

$$\frac{\Gamma \vdash \check{e} \Rightarrow \tau' \mid C \qquad \tau \nsim \tau'}{\Gamma \vdash (\!|\check{e}|\!)^{u}_{\nsim} \Leftarrow \tau \mid C \cup \{\tau \approx ?^{exp(u)}\}}\ \textsc{MAInconsistentTypes-C}$$

Unification with inconsistencies

For each occurrence of ?, accumulate the PotentialTypeSet

Unification with inconsistencies

For each occurrence of $?$, accumulate the PotentialTypeSet:
all potential type fillings as inferred from the constraints

$$\begin{array}{rrcl} \text{PotentialTypeSet} & s & ::= & \{t^*\} \\ \text{PotentialType} & t & ::= & ?^p \mid \text{num} \mid \text{bool} \mid s \rightarrow s \end{array}$$

Unification with inconsistencies

For each occurrence of $?$, accumulate the PotentialTypeSet: *all* potential type fillings as inferred from the constraints

$$\begin{array}{rrcl} \text{PotentialTypeSet} & s & ::= & \{t^*\} \\ \text{PotentialType} & t & ::= & ?^p \mid \text{num} \mid \text{bool} \mid s \rightarrow s \end{array}$$

To unify $\tau_1 \approx \tau_2$, recursively merge PotentialTypeSet(τ_1) and PotentialTypeSet(τ_2) without substituting

Unification with inconsistencies

$$\lambda f : ?^{1} \, . \; f \; (f + 1)$$

Unification with inconsistencies

$$\lambda f : ?^{1} .\ f\ (f + 1)$$

yields the (inconsistent) constraints

$$\{?^{1} \approx ?^{\to_L(1)} \to ?^{\to_R(1)}\}$$

Unification with inconsistencies

$$\lambda f : ?^{1} \,.\, f\ (f + 1)$$

yields the (inconsistent) constraints

$$\{?^{1} \approx ?^{\to_L(1)} \to ?^{\to_R(1)}, \mathbf{num} \approx ?^{1}\}$$

Unification with inconsistencies

$$\lambda f : ?^{1} \,.\, f\ (f + 1)$$

yields the (inconsistent) constraints

$$\{?^{1} \approx ?^{\to_L(1)} \to ?^{\to_R(1)}, \mathbf{num} \approx ?^{1}, \mathbf{num} \approx ?^{\to_L(1)}\}$$

Unification with inconsistencies

$$\lambda f : ?^{1} . \ f \ (f + 1)$$

yields the (inconsistent) constraints

$$\{?^{1} \approx ?^{\to_L(1)} \to ?^{\to_R(1)}, \mathbf{num} \approx ?^{1}, \mathbf{num} \approx ?^{\to_L(1)}\}$$

and solving gives

$$\text{PotentialTypeSet}(?^{1}) = \{\}$$

Unification with inconsistencies

$$\lambda f : ?^{1} \,.\, f\ (f + 1)$$

yields the (inconsistent) constraints

$$\{?^{1} \approx \underset{\wedge}{?^{\to_L(1)}} \to ?^{\to_R(1)}, \mathbf{num} \approx ?^{1}, \mathbf{num} \approx ?^{\to_L(1)}\}$$

and solving gives

$$\text{PotentialTypeSet}(?^{1}) = \{\underline{\{?^{\to_L(1)}\} \to \{?^{\to_R(1)}\}}\}$$

Unification with inconsistencies

$$\lambda f : ?^{1} \,.\, f\ (f + 1)$$

yields the (inconsistent) constraints

$$\{?^{1} \approx ?^{\to_L(1)} \to ?^{\to_R(1)}, \mathrm{num} \underset{\wedge}{\approx} ?^{1}, \mathrm{num} \approx ?^{\to_L(1)}\}$$

and solving gives

$$\mathrm{PotentialTypeSet}(?^{1}) = \{\{?^{\to_L(1)}\} \to \{?^{\to_R(1)}\}, \underline{\mathrm{num}}\}$$

Unification with inconsistencies

$$\lambda f : ?^{1} \, . \, f \; (f + 1)$$

yields the (inconsistent) constraints

$$\{?^{1} \approx ?^{\to_L(1)} \to ?^{\to_R(1)}, \mathbf{num} \approx ?^{1}, \mathbf{num} \underset{\wedge}{\approx} ?^{\to_L(1)}\}$$

and solving gives

$$\mathrm{PotentialTypeSet}(?^{1}) = \{\{\underline{\mathbf{num}}\} \to \{?^{\to_R(1)}\}, \mathbf{num}\}$$