Gradual Typing with Inference

Jeremy Siek
University of Colorado at Boulder

joint work with Manish Vachharajani

Overview

- Motivation
- Background
  - Gradual Typing
  - Unification-based inference
- Exploring the Solution Space
- Type system (specification)
- Inference algorithm (implementation)

Why Gradual Typing?

- Static and dynamic type systems have complimentary strengths.
- Static typing provides full-coverage error checking, efficient execution, and machine-checked documentation.
- Dynamic typing enables rapid development and fast adaption to changing requirements.
- Why not have both in the same language?

Java

Python

Goals for gradual typing

- Treat programs without type annotations as dynamically typed.
- Programmers may incrementally add type annotations to gradually increase static checking.
- Annotate all parameters and the type system catches all type errors.

Goals for gradual typing

- Treat programs without type annotations as dynamically typed.
- Programmers may incrementally add type annotations to gradually increase static checking.
- Annotate all parameters and the type system catches all type errors.

dynamic — static

Goals for gradual typing

- Treat programs without type annotations as dynamically typed.
- Programmers may incrementally add type annotations to gradually increase static checking.
- Annotate all parameters and the type system catches all type errors.

The Gradual Type System

- Classify dynamically typed expressions with the type '?'
- Allow implicit coercions *to* ? and *from* ? with any other type
- Extend coercions to compound types using a new *consistency relation*

Coercions to and from ‘?’

$(\lambda a{:}int.\ (\lambda x.\ x + 1)\ a)\ 1$

Parameters with no type annotation are given the dynamic type ‘?’.

Coercions to and from '?'

?

(λa:int. (λx. x + 1) a) 1

Parameters with no type annotation are given the dynamic type '?'.

Coercions to and from ‘?’

? int

(λa:int. (λx. x + 1) a) 1

Parameters with no type annotation are given the dynamic type ‘?’.

Coercions to and from '?'

? int int ⇒ ?

(λa:int. (λx. x + 1) a) 1

Parameters with no type annotation are given the dynamic type '?'.

Coercions to and from ‘?’

$\text{int} \Rightarrow ?$

$(\lambda a{:}\text{int}.\ (\lambda x.\ x + 1)\ a)\ 1$

$\text{int} \times \text{int} \to \text{int}$

Parameters with no type annotation are given the dynamic type ‘?’.

Coercions to and from ‘?’

$$\texttt{int} \Rightarrow \texttt{?}$$

?

(λa:int. (λx. x + 1) a) 1

int x int → int

Parameters with no type annotation are given the dynamic type ‘?’.

Coercions to and from '?'

? int ⇒ ?

(λa:int. (λx. x + 1) a) 1

int x int → int ? ⇒ int

Parameters with no type annotation are given the dynamic type '?'.

Coercions between compound types

$(\lambda f{:}int \rightarrow int.\ \ f\ 1)\ (\lambda x.\ 1)$

Coercions between compound types

? → int

(λf:int→int. f 1) (λx. 1)

Coercions between compound types

$? \rightarrow int$

(λf:int→int. f 1) (λx. 1)

$? \rightarrow int \;\Rightarrow\; int \rightarrow int$

Detect static type errors

(λf:int→int. f 1) 1

~~int ⇒ int → int~~

Type system: replace = with ~

$$\frac{\Gamma \vdash e_1 : \sigma \to \tau \qquad \Gamma \vdash e_2 : \sigma' \qquad \sigma' \sim \sigma}{\Gamma \vdash e_1\ e_2 : \tau}$$

Type system: replace = with ~

$$\frac{\Gamma \vdash e_1 : \sigma \to \tau \qquad \Gamma \vdash e_2 : \sigma' \qquad \sigma' \sim \sigma}{\Gamma \vdash e_1\ e_2 : \tau}$$

The consistency relation

- Definition: a type is *consistent*, written ~, with another type when they are equal where they are both defined.
- Examples:

$\text{int} \sim \text{int}$ $\quad$ $\text{int} \nsim \text{bool}$ $\quad$ $? \sim \text{int}$ $\quad$ $\text{int} \sim ?$

$\text{int} \rightarrow ? \sim ? \rightarrow \text{bool}$ $\quad$ $? \rightarrow \text{bool} \nsim ? \rightarrow \text{int}$

The consistency relation

$$\boxed{\tau_1 \sim \tau_2}$$

$$\frac{}{? \sim \tau} \qquad \frac{}{\tau \sim ?}$$

$$\frac{}{\tau \sim \tau}$$

$$\frac{\tau_1 \sim \tau_3 \qquad \tau_2 \sim \tau_4}{\tau_1 \rightarrow \tau_2 \sim \tau_3 \rightarrow \tau_4}$$

Compiler inserts run-time checks

$$\frac{\Gamma \vdash e_1 \Rightarrow e'_1 : \sigma \to \tau \qquad \Gamma \vdash e_2 \Rightarrow e'_2 : \sigma' \qquad \sigma' \sim \sigma}{\Gamma \vdash e_1\, e_2 \Rightarrow e'_1 \langle \sigma \Leftarrow \sigma' \rangle\, e'_2 : \tau}$$

Example:

$(\lambda a{:}int.\ (\lambda x.\ x + 1)\ a)\ 1$

$\Rightarrow$

$(\lambda a{:}int.\ (\lambda x.\ \langle int \Leftarrow ? \rangle x + 1)\ \langle ? \Leftarrow int \rangle a)\ 1$

Recent Developments

- Integration with objects (Siek & Taha, ECOOP'07)
- Space-efficiency (Herman et al, TFP'07)
- Blame tracking (Wadler & Findler, Scheme'07)
- In JavaScript (Herman & Flanagan, ML'07)

Why Inference?

- Interesting research question: how does the dynamic type interact with type variables?
- Practical applications
  - Help programmers migrate dynamically typed code to statically typed code
  - Explain how gradual typing can be integrated with functional languages with inference (ML, Haskell, etc.)

STLC with type vars: Specification

Standard STLC judgment: $\Gamma \vdash e : \tau$

An STLC term with type variables is *well typed* if there exists an S such that

$$S(\Gamma) \vdash S(e) : S(\tau)$$

e.g., $(\lambda x{:}\text{int}.\ (\lambda y{:}\alpha.\ y)\ x)$

$$S = \{\alpha \mapsto \text{int}\}$$

Inference Algorithm

$$\lambda x{:}\text{int}.\ (\lambda y{:}\alpha.\ y)\ x$$

↓ constraint generation

$$\alpha \to \alpha = \text{int} \to \beta$$

↓ unification

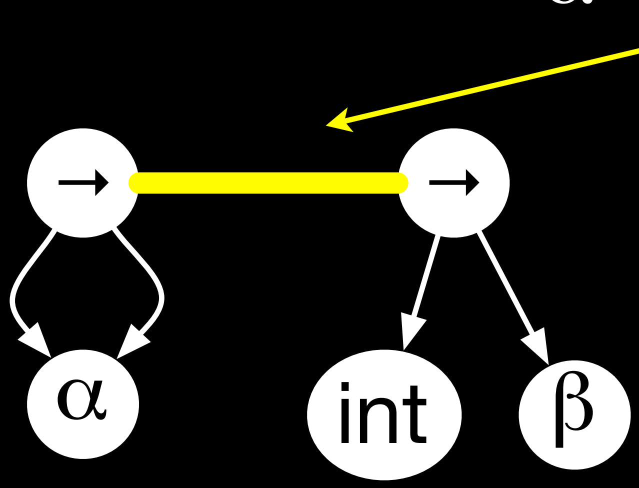
$$S = \{\alpha \mapsto \text{int},\ \beta \mapsto \text{int}\}$$

Huet's Unification

$$\alpha \to \alpha = \text{int} \to \beta$$

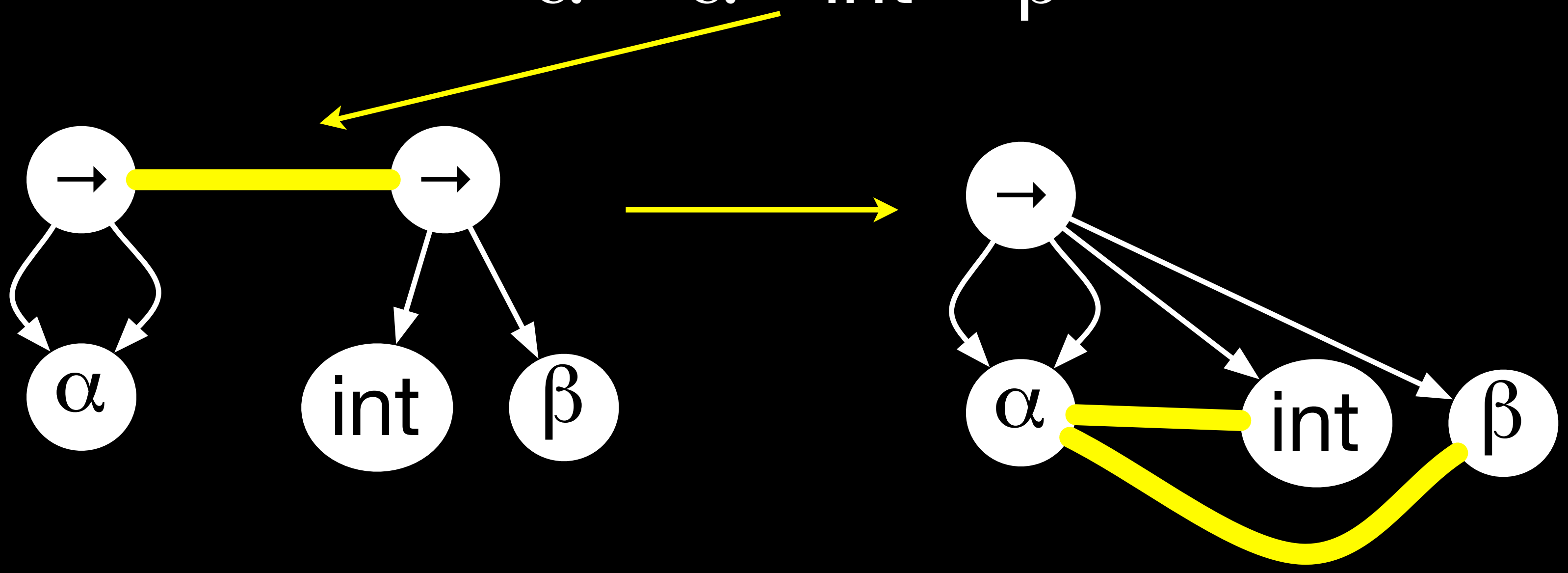
Huet's Unification

$\alpha \to \alpha = \text{int} \to \beta$



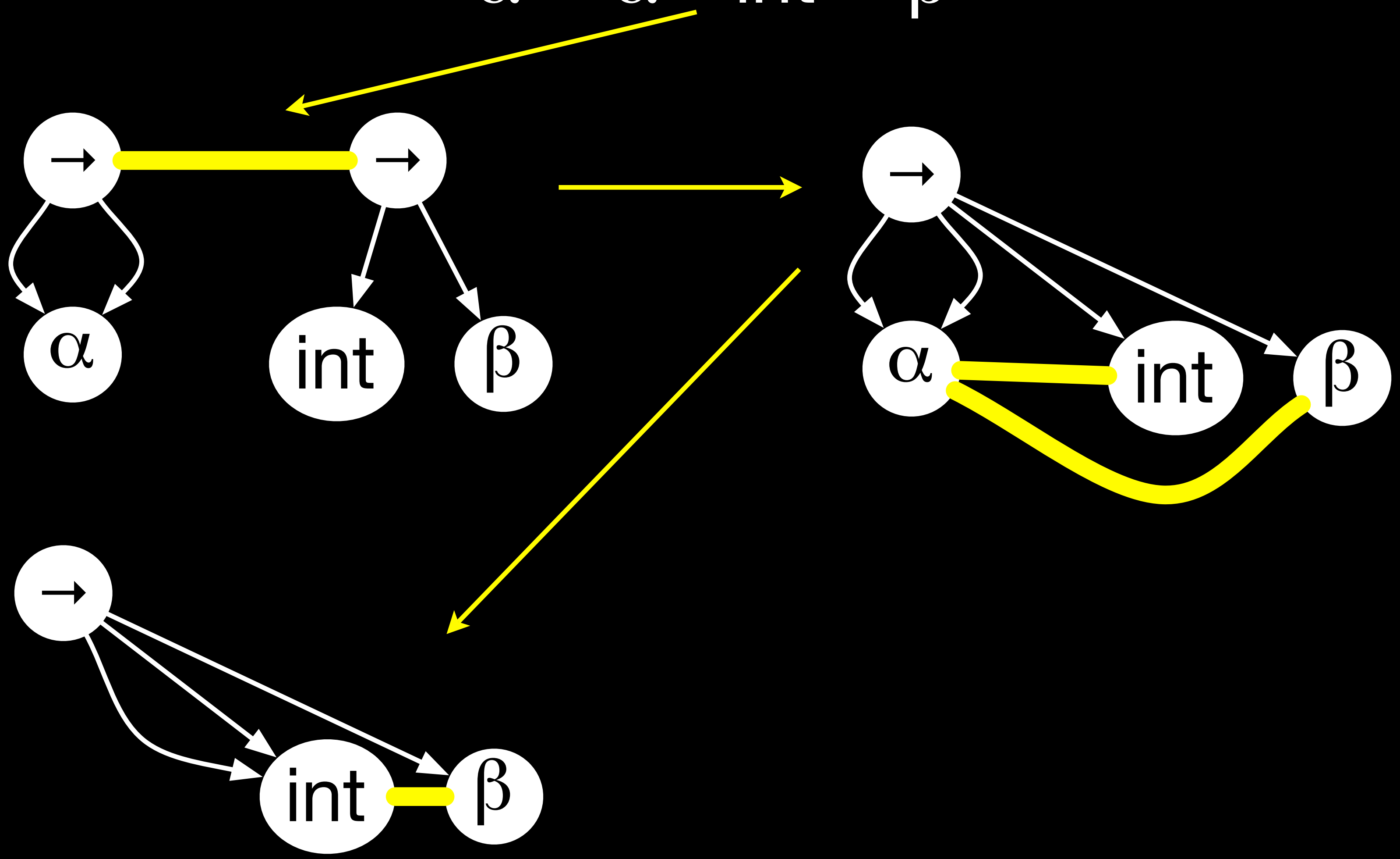
Huet's Unification

$\alpha \to \alpha = \text{int} \to \beta$



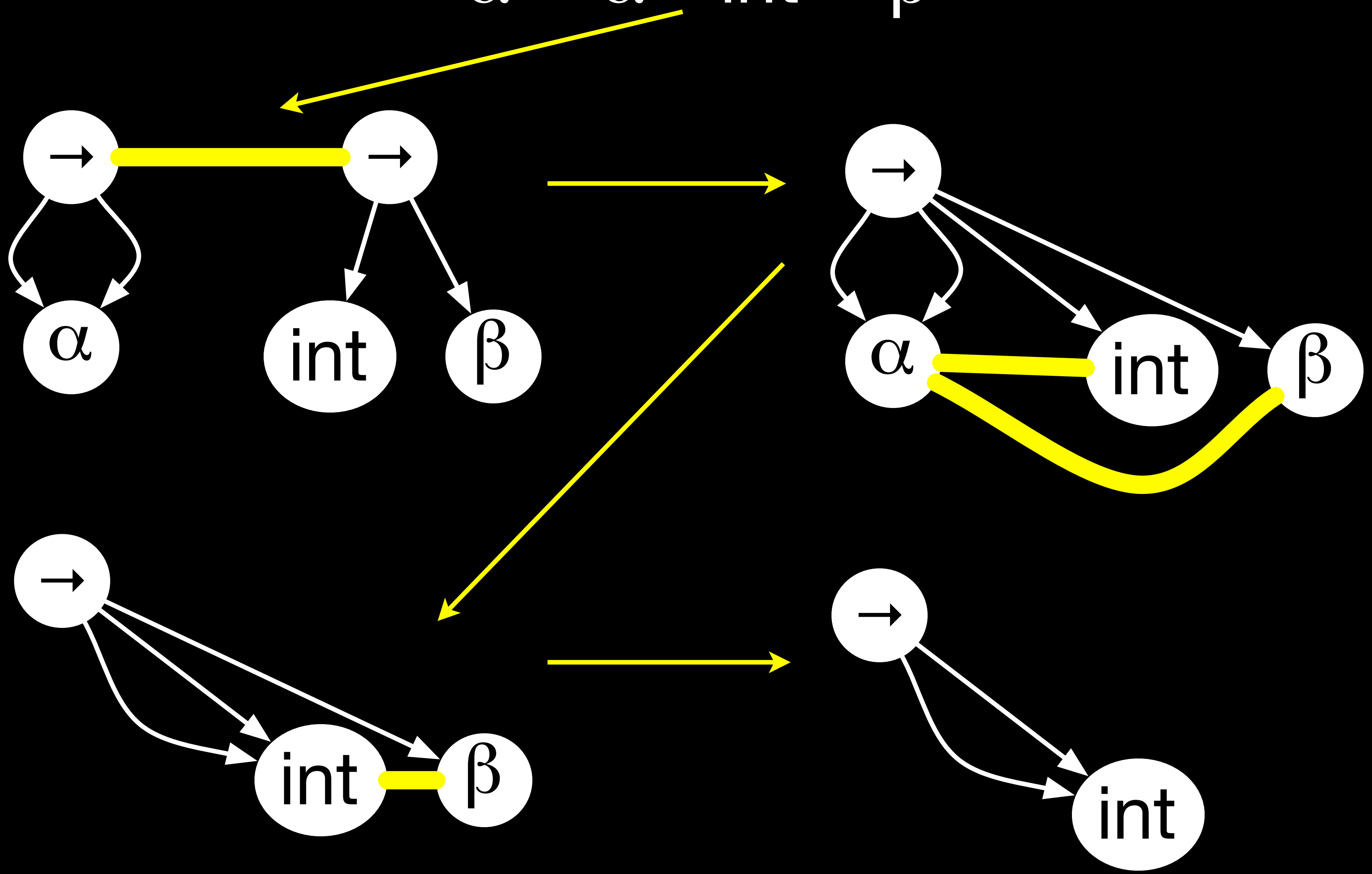
Huet's Unification

$\alpha \to \alpha = \text{int} \to \beta$



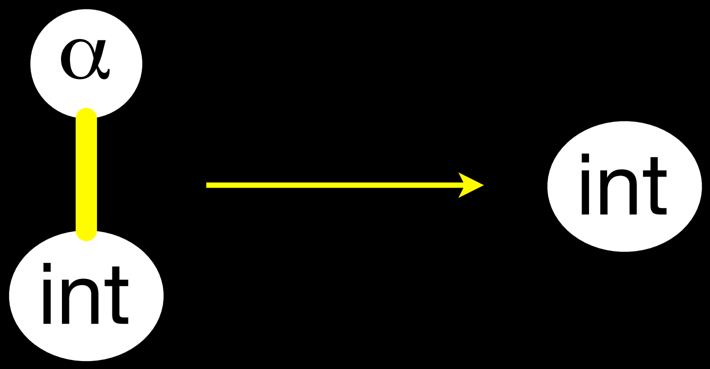
Huet's Unification

$\alpha \to \alpha = \text{int} \to \beta$



Huet's Unification

- When merging nodes, the algorithm needs to decide which label to keep
- In this setting, non-type variables trump type variables



Gradual Typing with Inference

- Setting: STLC with α and ?.
- To migrate from dynamic to static, change ? to α and the inferencer will tell you the solution for α or give an error.

$$\lambda\ f{:}?.\ \lambda\ x{:}?.\ f\ x\ x$$

$$\downarrow$$

$$\lambda\ f{:}\alpha.\ \lambda\ x{:}?.\ f\ x\ x$$

Syntactic Sugar

λ f. λ x. f x x

?

Syntactic Sugar

λ f. λ x. f x x

λ f:?. λ x:?. f x x

?

Syntactic Sugar

λ f. λ x. f x x

λ f:?. λ x:?. f x x

λ f:α. λ x:β. f x x

?

Non-solution #1

Well typed in gradual type system
after substitution

$$S(\Gamma) \vdash S(e) : S(\tau)$$

Problem: the following is accepted

$$(\lambda\ f{:}\alpha.\ f\ 1)\ 1$$

$$S = \{\alpha \mapsto ?\}$$

Non-solution #2

Forbid ?s from appearing in a solution S

Problem: sometimes this forces cast errors at runtime

λx:?. (λ y:α. y) x

λx:?. (λ y:int. y) x

λx:?. (λ y:int. y) $\langle$int$\Leftarrow$?$\rangle$x

Non-solution #2

Forbid ?s from appearing in a solution S

Problem: sometimes this forces cast errors at runtime

$\lambda x{:}?.\ (\lambda\ y{:}\alpha.\ y)\ x \longrightarrow \lambda x{:}?.\ (\lambda\ y{:}\mathrm{int}.\ y)\ x$

$\downarrow$

$\lambda x{:}?.\ (\lambda\ y{:}\mathrm{int}.\ y)\ \langle \mathrm{int} \Leftarrow ? \rangle x$

Non-solution #3

Treat each ? as a different type variable
then check for well typed in STLC after substitution

Problem: the following is rejected

$$\lambda\ f{:}int \to bool \to int.\ \lambda\ x{:}?.\ f\ x\ x$$

↓

$$\lambda\ f{:}int \to bool \to int.\ \lambda\ x{:}\alpha.\ f\ x\ x$$

Non-solution #4

Treat each occurrence of ? in a constraint as a different type variable

Problem: if no type vars in the program, the resulting type should not have type vars

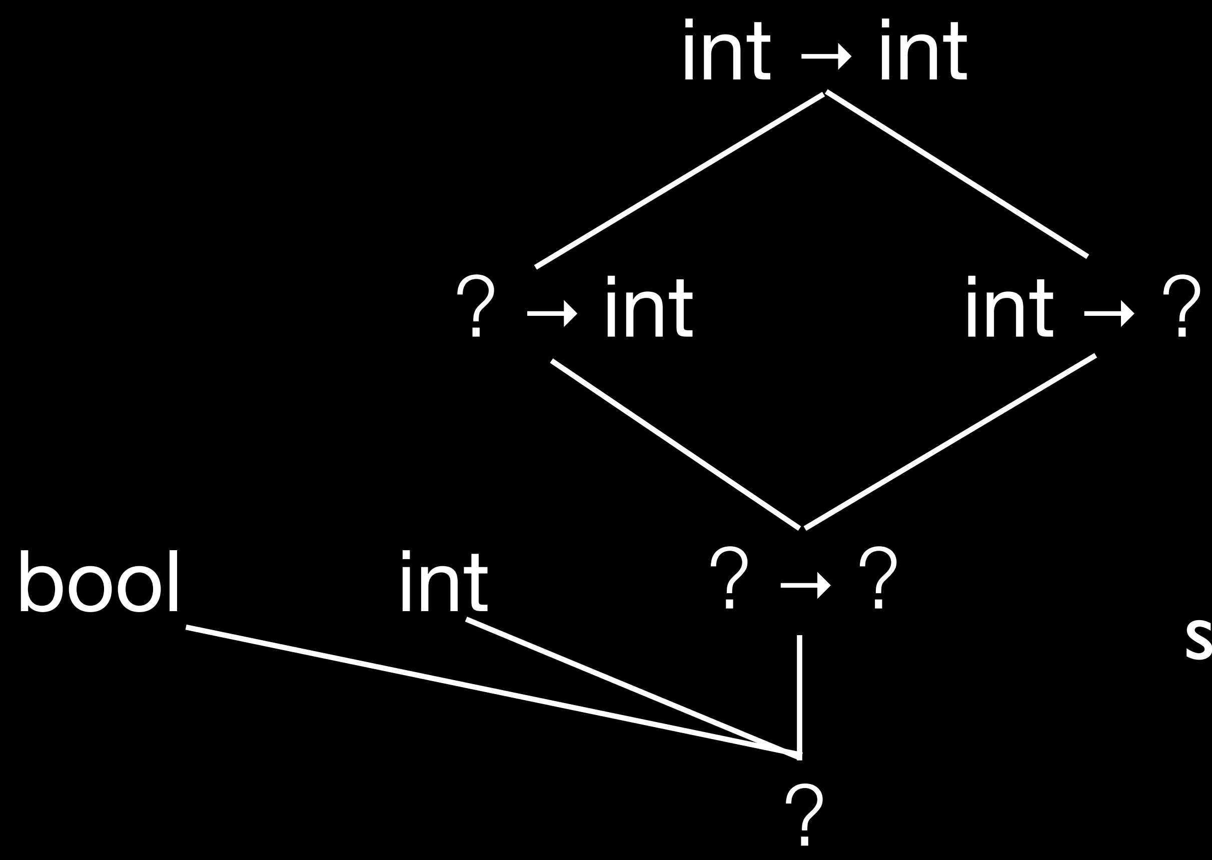
λ f:int → ?. λ x:int. (f x)

int → ? = int → β ⟶ int → α = int → β

Lessons

- Need to restrict the occurrences of ? in solutions
- But can't completely outlaw the use of ?
- Idea: a solution for α at least as informative as any of the types that constrain α constrain
- i.e., the solution for α must be an upper bound of all the types that constrain α

Information Ordering



$\tau_1 \sqsubseteq \tau_2$

semi-lattice

Type System

- But what does it mean for a type to constrain α?

$$\lambda f{:}\alpha \to \alpha.\ \lambda g{:}(? \to \text{int}) \to \text{int}.\ g\ f$$

$\alpha \to \alpha$ $\quad\quad$ $? \to \text{int}$

Type System

- But what does it mean for a type to constrain α?

$$\lambda f{:}\alpha \to \alpha.\ \lambda g{:}(? \to \text{int}) \to \text{int}.\ g\ f$$

$$\alpha \to \alpha \qquad ? \to \text{int}$$

$$? \sqsubseteq S(\alpha)$$

Type System

- But what does it mean for a type to constrain α?

$$\lambda f{:}\alpha\to\alpha.\ \lambda g{:}(?\to int)\to int.\ g\ f$$

$$\alpha \to \alpha \qquad ? \to int$$

$$? \sqsubseteq S(\alpha)$$

$$int \sqsubseteq S(\alpha)$$

Type System

- The typing judgment: $S; \Gamma \vdash e : \tau$
- Consistent-equal: $S \models \tau \simeq \tau$
- Consistent-less: $S \models \tau \sqsubseteq \tau$

Type System

$$S; \Gamma \vdash e : \tau$$

$$\frac{S; \Gamma \vdash e_1 : \tau_1 \qquad S; \Gamma \vdash e_2 : \tau_2 \qquad S \models \tau_1 \simeq \tau_2 \to \beta \qquad (\beta \text{ fresh})}{S; \Gamma \vdash e_1\ e_2 : \beta}$$

Type System

$$\boxed{S; \Gamma \vdash e : \tau}$$

$$\frac{S; \Gamma \vdash e_1 : \tau_1 \qquad S; \Gamma \vdash e_2 : \tau_2 \qquad S \models \tau_1 \simeq \tau_2 \to \beta \quad (\beta \text{ fresh})}{S; \Gamma \vdash e_1\ e_2 : \beta}$$

Consistent-equal

$$\boxed{S \models \tau \simeq \tau}$$

$$\frac{}{S \models ? \simeq \tau} \qquad \frac{}{S \models \tau \simeq ?}$$

$$\frac{S \models \tau \sqsubseteq S(\alpha)}{S \models \alpha \simeq \tau} \qquad \frac{S \models \tau \sqsubseteq S(\alpha)}{S \models \tau \simeq \alpha}$$

$$\frac{}{S \models \gamma \simeq \gamma} \qquad \frac{S \models \tau_1 \simeq \tau_3 \qquad S \models \tau_2 \simeq \tau_4}{S \models \tau_1 \rightarrow \tau_2 \simeq \tau_3 \rightarrow \tau_4}$$

Consistent-less

$$S \models \tau \sqsubseteq \tau$$

$$S \models ? \sqsubseteq \tau$$

$$\frac{S \models S(\alpha) = \tau}{S \models \alpha \sqsubseteq \tau}$$

$$\frac{}{S \models \gamma \sqsubseteq \gamma}$$

$$\frac{S \models \tau_1 \sqsubseteq \tau_3 \qquad S \models \tau_2 \sqsubseteq \tau_4}{S \models \tau_1 \rightarrow \tau_2 \sqsubseteq \tau_3 \rightarrow \tau_4}$$

Properties

- When there are no type variables in the program, the type system acts like the original gradual type system
- When there are no ? in the program, the type system acts like the STLC with variables

Inference Algorithm

$\lambda f{:}\alpha\to\alpha.\ \lambda g{:}(?\to int)\to int.\ g\ f$

↓ constraint generation

$(? \to int) \to int \ \simeq (\alpha \to \alpha) \to \beta$

↓ unification for ≃

$S = \{\alpha \mapsto int,\ \beta \mapsto int\}$

Unification for $\simeq$

- Can't use the standard substitution-based version because we need to see all the unificands before deciding on the solution

$$(? \rightarrow \text{int}) \rightarrow \text{int} \simeq (\alpha \rightarrow \alpha) \rightarrow \beta$$

Unification for $\simeq$

- Need to compute the *least* upper bound
- Otherwise spurious casts are inserted

$\lambda x{:}?.\ (\lambda\ y{:}\alpha.\ y)\ x$

$\lambda x{:}?.\ (\lambda\ y{:}\mathrm{int}.\ y)\ x$

$\lambda x{:}?.\ (\lambda\ y{:}\mathrm{int}.\ y)\ \langle \mathrm{int} \Leftarrow ? \rangle x$

Unification for $\simeq$

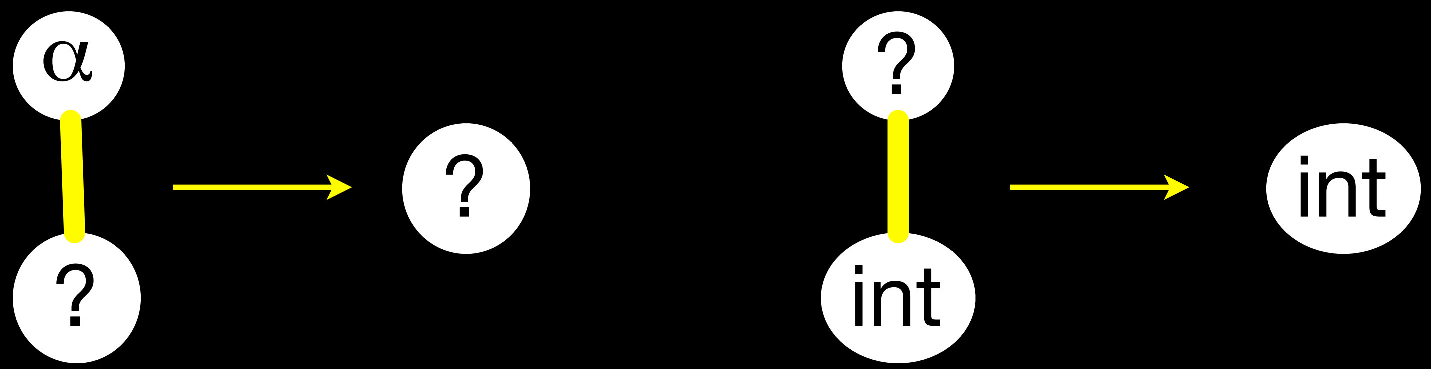
- Need to compute the *least* upper bound
- Otherwise spurious casts are inserted

$$\lambda x{:}?.\ (\lambda\ y{:}\alpha.\ y)\ x \longrightarrow \lambda x{:}?.\ (\lambda\ y{:}\mathrm{int}.\ y)\ x$$

$$\lambda x{:}?.\ (\lambda\ y{:}\mathrm{int}.\ y)\ \langle \mathrm{int} \Leftarrow ? \rangle x$$

Merging Labels

- Type variables are trumped by non-type variables (including the dynamic type)
- The dynamic type is trumped by concrete types (e.g., int, bool, →)

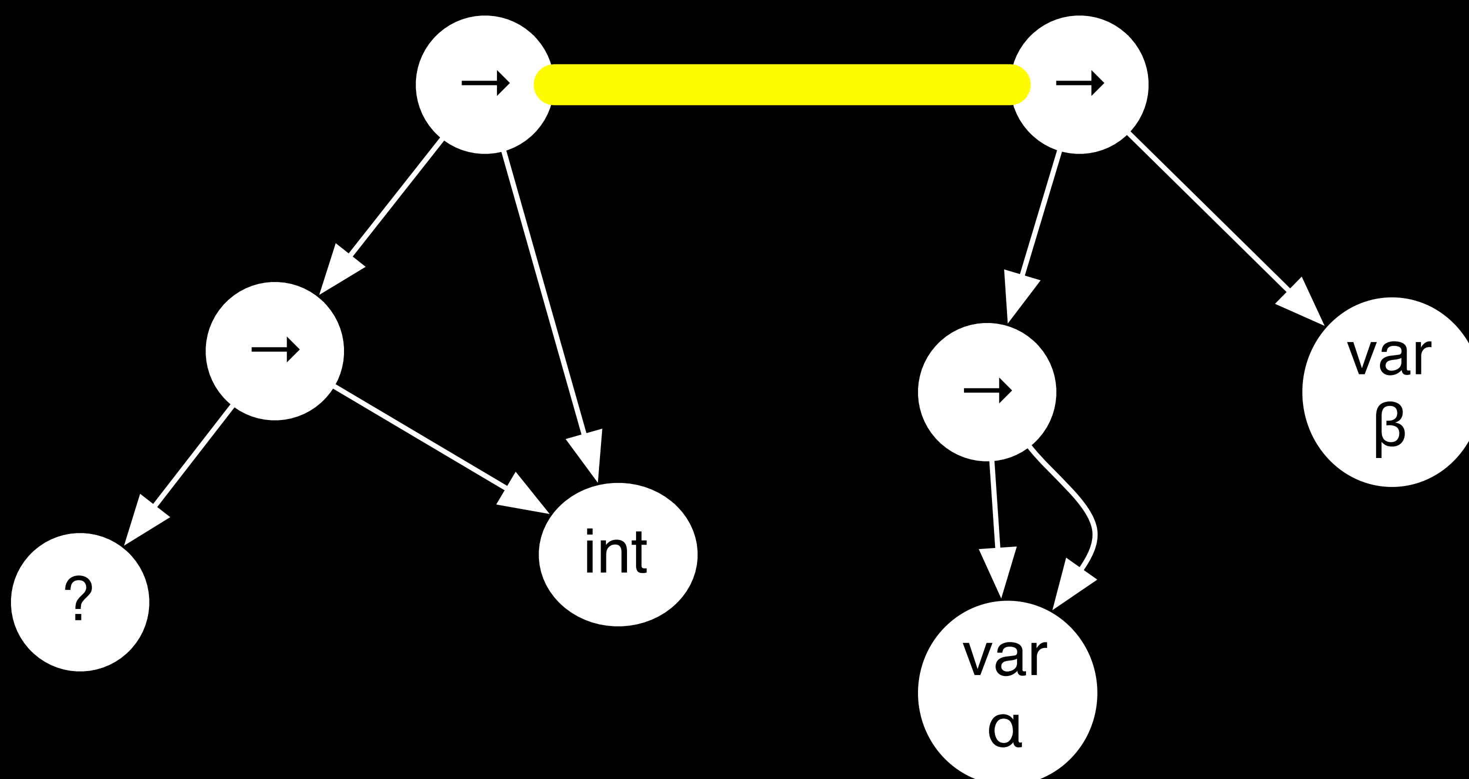


Unification for $\simeq$

$$(? \to \text{int}) \to \text{int} \simeq (\alpha \to \alpha) \to \beta$$

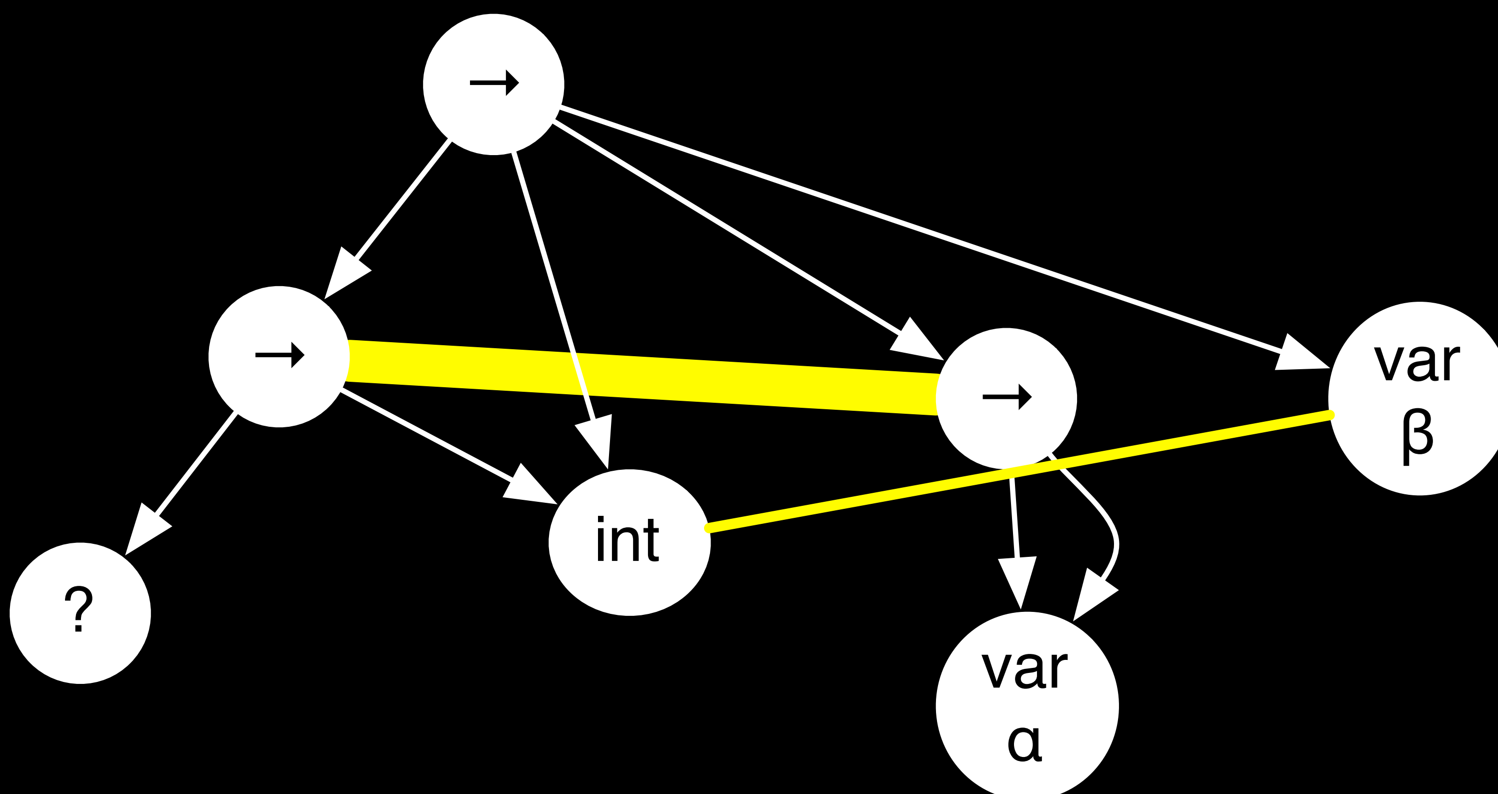
Unification for $\simeq$

$(? \to \text{int}) \to \text{int} \simeq (\alpha \to \alpha) \to \beta$



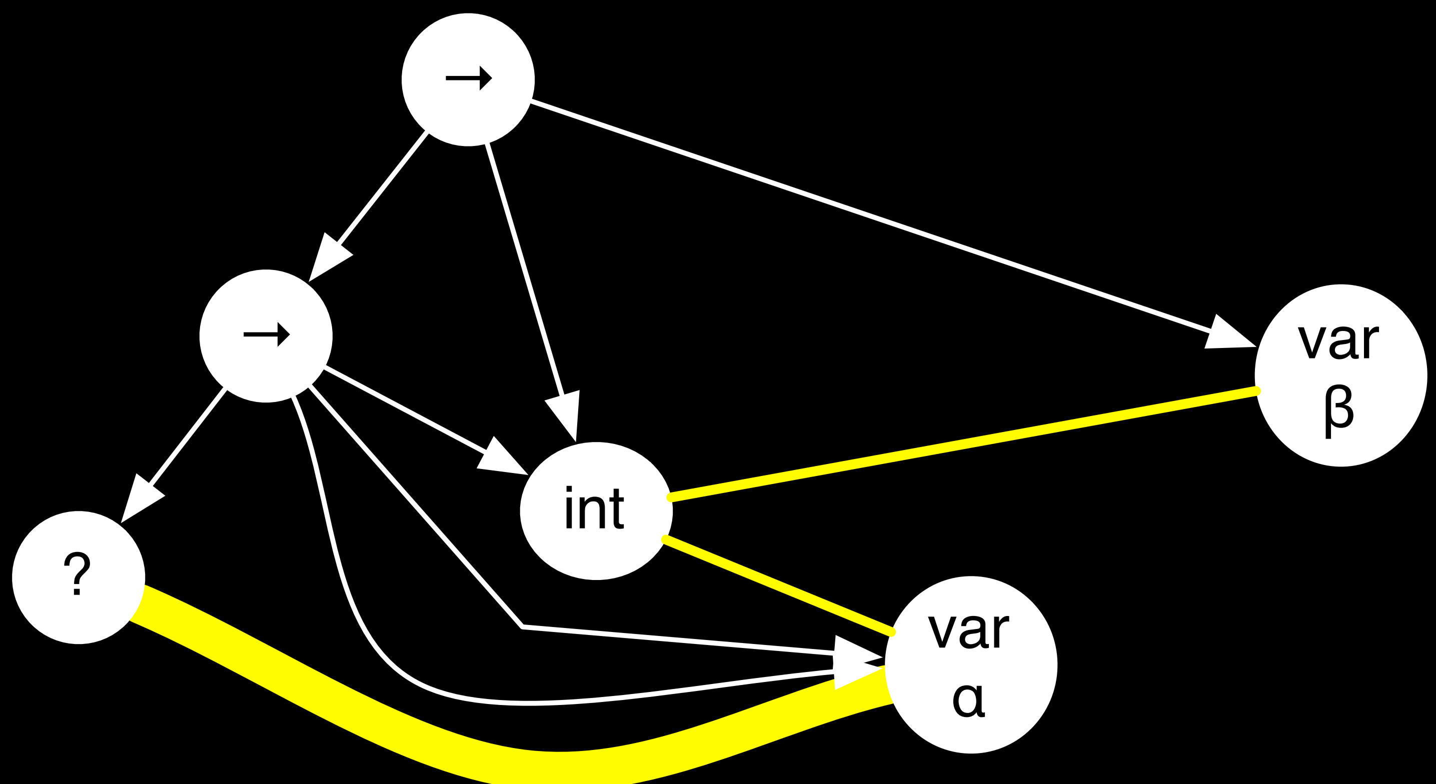
Unification for ≃

$(? \to \text{int}) \to \text{int} \simeq (\alpha \to \alpha) \to \beta$



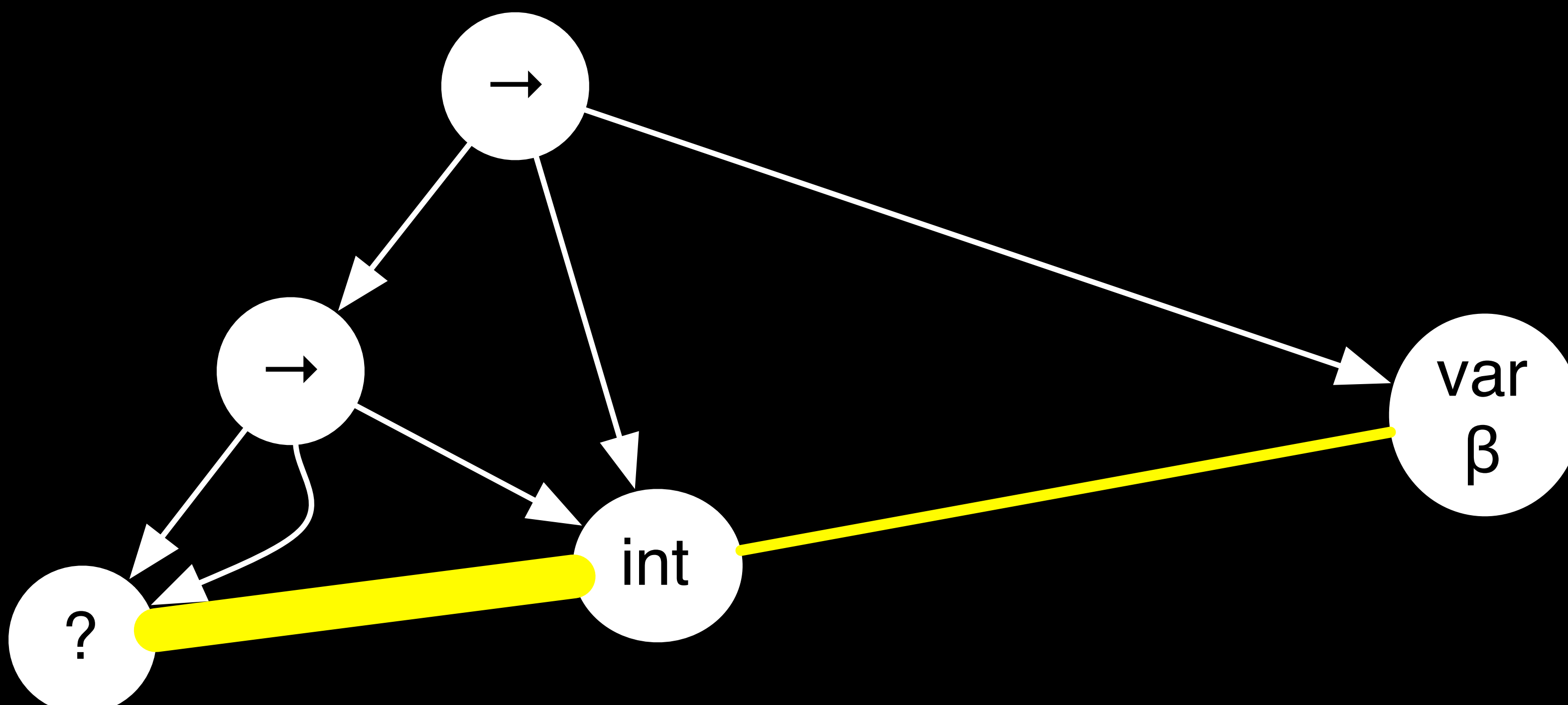
Unification for ≃

$$(? \to \text{int}) \to \text{int} \simeq (\alpha \to \alpha) \to \beta$$



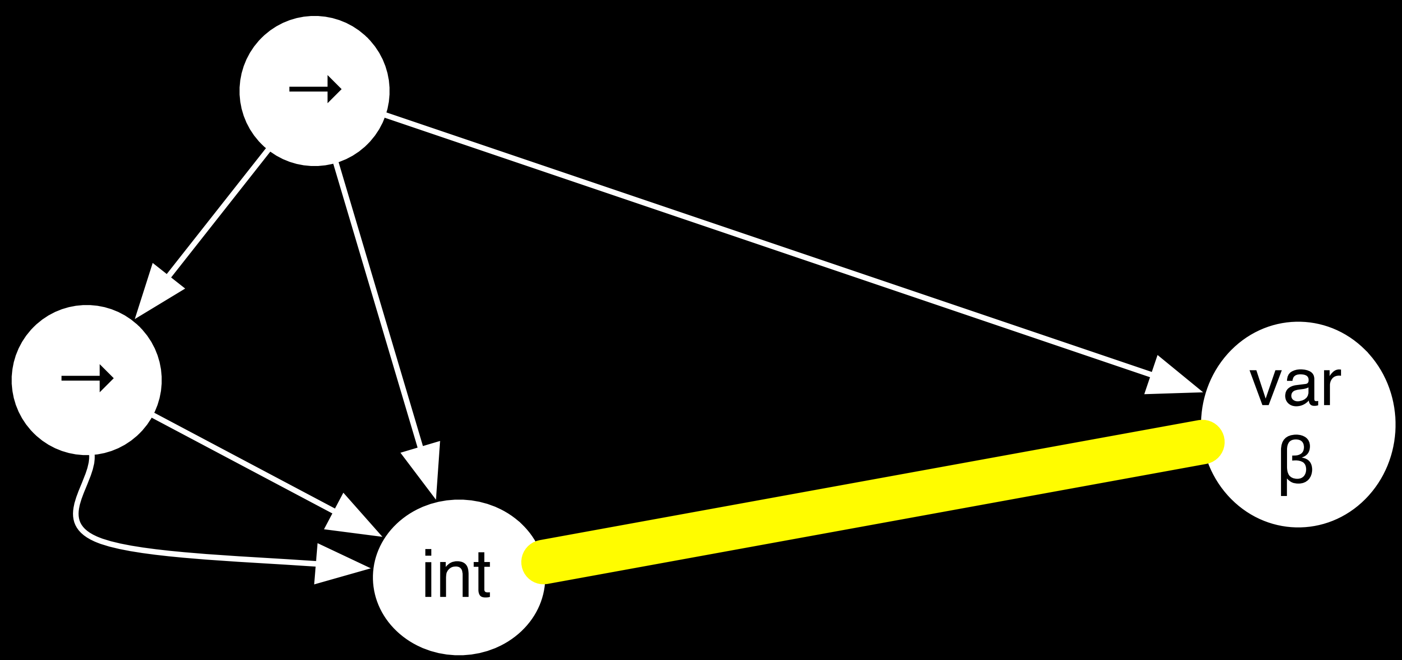
Unification for ≃

$$(? \to \text{int}) \to \text{int} \simeq (\alpha \to \alpha) \to \beta$$



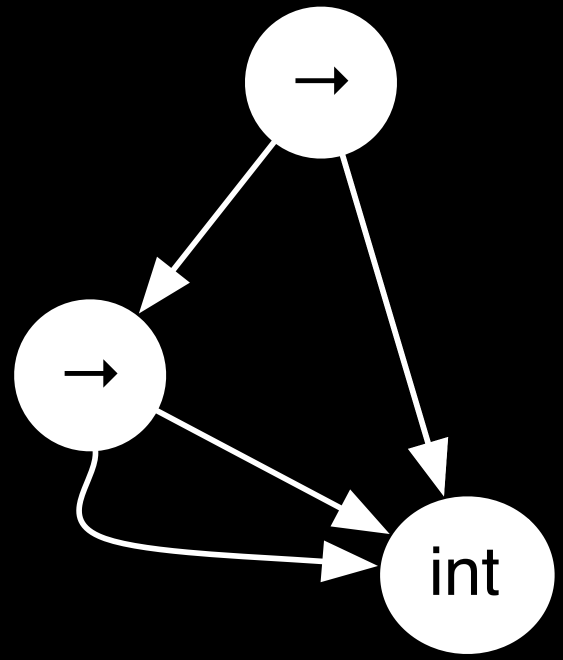
Unification for ≃

$(? \to \text{int}) \to \text{int} \simeq (\alpha \to \alpha) \to \beta$



Unification for ≃

$(? \to \text{int}) \to \text{int} \simeq (\alpha \to \alpha) \to \beta$



Properties

- The time complexity of unification for $\simeq$ is $O(m\ \alpha(n))$ for a graph with n nodes and m edges
- Soundness: if $(S,\tau) = \text{infer}(\Gamma, e)$ then $S^*; \Gamma \vdash e : \tau$.
- Completeness: if $S; \Gamma \vdash e : \tau$ then there is a S', τ', and R such that $(S', \tau') = \text{infer}(\Gamma, e)$ and $R \bullet S' \sqsubseteq S$ and $R \bullet S'^*(\tau') \sqsubseteq S(\tau)$.

Related Work

- Java + Dynamic (Gray & Findler & Flatt)
- Optional types (LISP, Dylan, etc.)
- BabyJ: gradual typing in a nominal setting(Anderson & Drossopoulou)
- Quasi-static types (Thatte)
- Soft typing (Cartwright & Fagan, Wright & Cartwright, Flanagan & Felleisen, Aiken & Wimmers & Lakshman)
- Dynamic typing (Henglein)

Conclusion

- Gradual typing provides a combination of dynamic and static typing in the same language, under programmer control.
- We present a type system for gradually typed programs with type variables.
- We present a unification-based inference algorithm that only requires a small change to Huet's algorithm to handle ?s.

Type System

$$\frac{S; \Gamma \vdash e_1 : \tau_1 \quad S; \Gamma \vdash e_2 : \tau_2 \quad S \models \tau_1 \simeq \tau_2 \to \beta \quad (\beta \text{ fresh})}{S; \Gamma \vdash e_1\ e_2 : \beta}$$

Type System

$$\frac{S;\Gamma \vdash e_1 : \tau_1 \quad S;\Gamma \vdash e_2 : \tau_2 \quad S \models \tau_1 \simeq \tau_2 \to \beta \quad (\beta \text{ fresh})}{S;\Gamma \vdash e_1\ e_2 : \beta}$$

Non-solution

$$\frac{S; \Gamma \vdash e_1 : \tau_1 \qquad S; \Gamma \vdash e_2 : \tau_2 \qquad S \models \tau_1 \simeq \tau_2 \to \tau_3}{S; \Gamma \vdash e_1\ e_2 : \tau_3}$$

Problem: the following is accepted because we can choose $\tau_3 = ?$

λ f:int → int. λ g:int → bool. f (g 1)

Solution

$$\frac{S; \Gamma \vdash e_1 : \tau_1 \quad S; \Gamma \vdash e_2 : \tau_2 \qquad S \vDash \tau_1 \simeq \tau_2 \to \beta \quad (\beta \text{ fresh})}{S; \Gamma \vdash e_1\ e_2 : \beta}$$

λ f:int → int. λ g:int → bool. f (g 1)

$S \vDash \text{int} \to \text{bool} \simeq \text{int} \to \beta_1$

$S \vDash \text{int} \to \text{int} \simeq \beta_1 \to \beta_2$

→

$S \vDash \text{bool} \sqsubseteq \beta_1$

$S \vDash \text{int} \sqsubseteq \beta_1$

Inference Algorithm

$\lambda f{:}(?\to int)\to(int\to ?)\to int.\ \lambda y{:}\alpha.\ f\ y\ y$

↓ constraint generation

$$(?\to int)\to (int\to ?)\to int \simeq \alpha \to \beta_1$$

$$\beta_1 \simeq \alpha \to \beta_2$$

↓ unification for $\simeq$

$S = \{\alpha \mapsto int\to int,\ \beta_1 \mapsto (int\to int)\to int,\ \beta_2 \mapsto int\}$

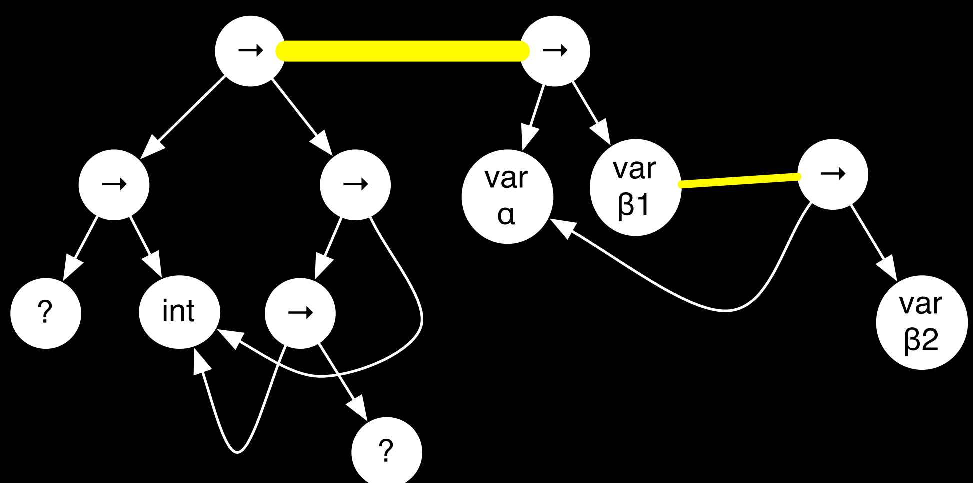
Unification for $\simeq$

$$(? \to \text{int}) \to (\text{int} \to ?) \to \text{int} \simeq \alpha \to \beta_1$$

$$\beta_1 \simeq \alpha \to \beta_2$$

Unification for $\simeq$

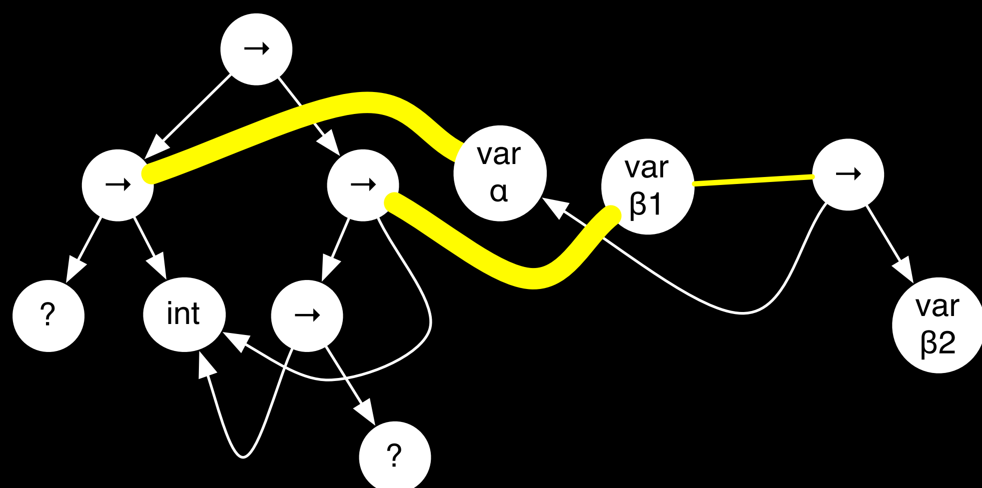
$$(?\to\text{int}) \to (\text{int}\to?)\to\text{int} \simeq \alpha \to \beta_1$$
$$\beta_1 \simeq \alpha \to \beta_2$$



Unification for ≃

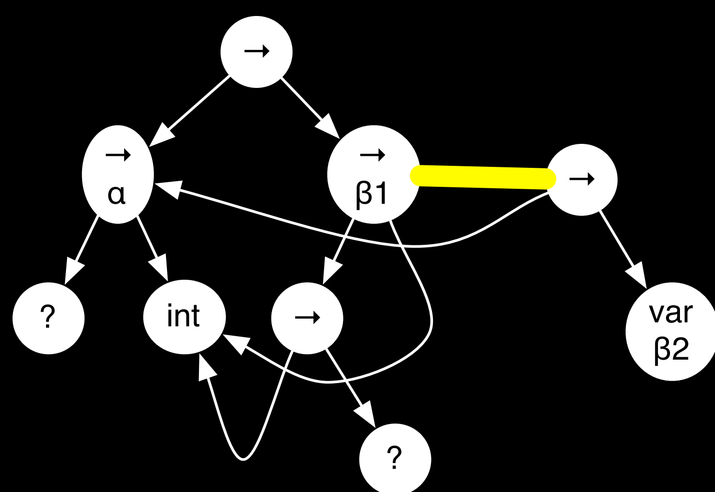
$$(? \to \text{int}) \to (\text{int} \to ?) \to \text{int} \simeq \alpha \to \beta_1$$

$$\beta_1 \simeq \alpha \to \beta_2$$



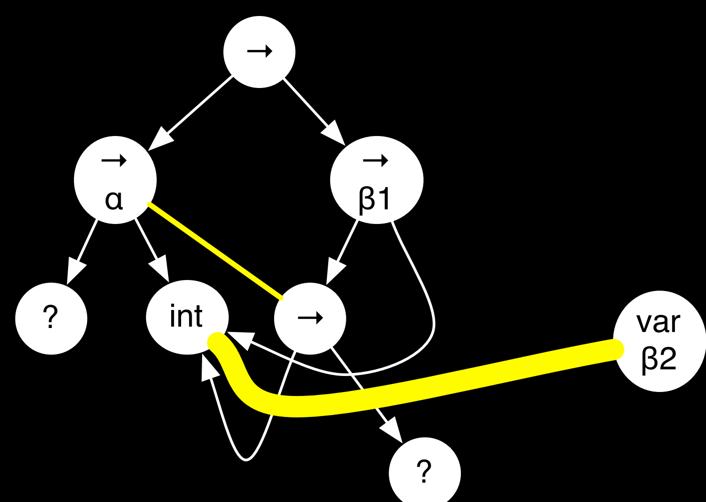
Unification for ≃

$$(? \to int) \to (int \to ?) \to int \simeq \alpha \to \beta_1$$
$$\beta_1 \simeq \alpha \to \beta_2$$



Unification for $\simeq$

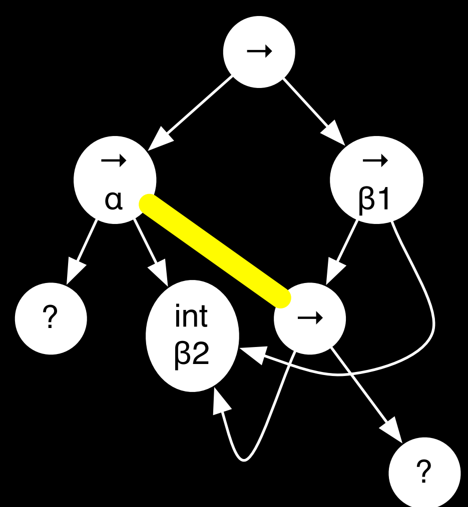
$$(? \to \text{int}) \to (\text{int} \to ?) \to \text{int} \simeq \alpha \to \beta_1$$
$$\beta_1 \simeq \alpha \to \beta_2$$



Unification for ≃

$$(?\to\text{int}) \to (\text{int}\to ?)\to\text{int} \simeq \alpha \to \beta_1$$

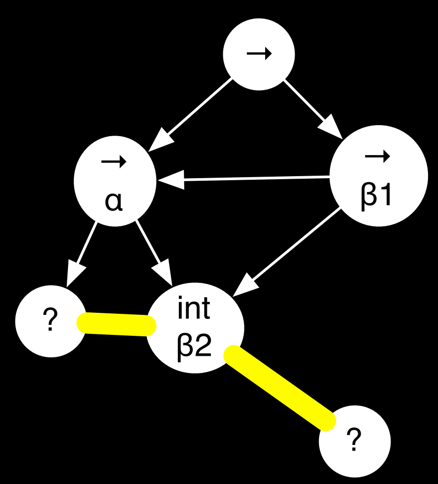
$$\beta_1 \simeq \alpha \to \beta_2$$



Unification for ≃

$$(? \to \text{int}) \to (\text{int} \to ?) \to \text{int} \simeq \alpha \to \beta_1$$

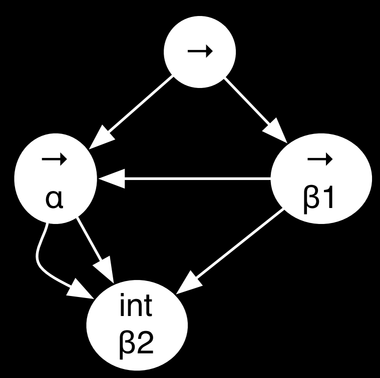
$$\beta_1 \simeq \alpha \to \beta_2$$



Unification for ≃

$$(? \to \text{int}) \to (\text{int} \to ?) \to \text{int} \simeq \alpha \to \beta_1$$

$$\beta_1 \simeq \alpha \to \beta_2$$



Unification for $\simeq$

$$(? \to \text{int}) \to (\text{int} \to ?) \to \text{int} \simeq \alpha \to \beta_1$$

$$\beta_1 \simeq \alpha \to \beta_2$$