Last time

System Fω

$$\frac{K_1 \text{ is a kind} \qquad K_2 \text{ is a kind}}{K_1 \Rightarrow K_2 \text{ is a kind}} \Rightarrow\text{-kind}$$

$$\frac{\Gamma, \alpha{::}K_1 \vdash A :: K_2}{\Gamma \vdash \lambda\alpha{::}K_1.A :: K_1 \Rightarrow K_2} \Rightarrow\text{-intro} \qquad \frac{\Gamma \vdash A :: K_1 \Rightarrow K_2 \quad \Gamma \vdash B :: K_1}{\Gamma \vdash A\ B :: K_2} \Rightarrow\text{-elim}$$

(and encoding data types: 1, 2, $\mathbb{N}$, $+$, lists, nested types and $\equiv$)

This time

$$\Gamma \vdash M : ?$$

What is type inference?

```
# fun f g x -> f (g x);;
- : ('a -> 'b) -> ('c -> 'a) -> 'c -> 'b = <fun>
```

What is type inference?

```
# fun f g x -> f (g x);;
- : ('a -> 'b) -> ('c -> 'a) -> 'c -> 'b = <fun>
```

Goal

succinctness of annotation-free code

$+$

safety and expressiveness of System Fω

What is type inference?

```
# fun f g x -> f (g x);;
- : ('a -> 'b) -> ('c -> 'a) -> 'c -> 'b = <fun>
```

Goal

succinctness of annotation-free code

$+$

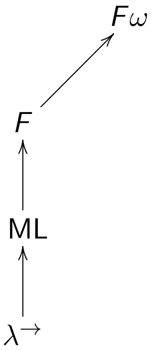
safety and expressiveness of System Fω

Bad news

the goal is unachievable

The ML calculus

The ML calculus



Prenex quantifification

$\forall\alpha.\alpha \to \alpha$

$\forall\alpha.\forall\beta.\alpha \to (\beta \to \beta)$

$\forall\alpha.(\forall\beta.\beta \to \beta) \to \alpha$

$\forall\alpha.\alpha \to (\forall\beta.\beta \to \beta)$

Let-bound polymorphism

```
let id = fun x -> x
 in id id
```

```
let id x = x
 in id id
```

```
let f id = id id
 in f (fun x -> x)
```

```
(fun id -> id id)
  (fun x -> x)
```

Prenex quantifification

$\forall \alpha . \alpha \to \alpha$ ✓

$\forall \alpha . \forall \beta . \alpha \to (\beta \to \beta)$

$\forall \alpha . (\forall \beta . \beta \to \beta) \to \alpha$

$\forall \alpha . \alpha \to (\forall \beta . \beta \to \beta)$

Let-bound polymorphism

```
let id = fun x -> x
 in id id
```

```
let id x = x
 in id id
```

```
let f id = id id
 in f (fun x -> x)
```

```
(fun id -> id id)
  (fun x -> x)
```

Prenex quantifification

$\forall\alpha.\alpha \to \alpha$ ✓

$\forall\alpha.\forall\beta.\alpha \to (\beta \to \beta)$ ✓

$\forall\alpha.(\forall\beta.\beta \to \beta) \to \alpha$

$\forall\alpha.\alpha \to (\forall\beta.\beta \to \beta)$

Let-bound polymorphism

```
let id = fun x -> x
 in id id
```

```
let id x = x
 in id id
```

```
let f id = id id
 in f (fun x -> x)
```

```
(fun id -> id id)
  (fun x -> x)
```

Prenex quantifification

$\forall\alpha.\alpha \to \alpha$ ✓

$\forall\alpha.\forall\beta.\alpha \to (\beta \to \beta)$ ✓

$\forall\alpha.(\forall\beta.\beta \to \beta) \to \alpha$ ✗

$\forall\alpha.\alpha \to (\forall\beta.\beta \to \beta)$

Let-bound polymorphism

```
let id = fun x -> x
 in id id
```

```
let id x = x
 in id id
```

```
let f id = id id
 in f (fun x -> x)
```

```
(fun id -> id id)
  (fun x -> x)
```

Prenex quantifification

$\forall\alpha.\alpha \to \alpha$ ✓

$\forall\alpha.\forall\beta.\alpha \to (\beta \to \beta)$ ✓

$\forall\alpha.(\forall\beta.\beta \to \beta) \to \alpha$ ✗

$\forall\alpha.\alpha \to (\forall\beta.\beta \to \beta)$ ✗

Let-bound polymorphism

```
let id = fun x -> x
 in id id
```

```
let id x = x
 in id id
```

```
let f id = id id
 in f (fun x -> x)
```

```
(fun id -> id id)
  (fun x -> x)
```

Prenex quantifification

$\forall\alpha.\alpha \to \alpha$ ✓

$\forall\alpha.\forall\beta.\alpha \to (\beta \to \beta)$ ✓

$\forall\alpha.(\forall\beta.\beta \to \beta) \to \alpha$ ✗

$\forall\alpha.\alpha \to (\forall\beta.\beta \to \beta)$ ✗

Let-bound polymorphism

```
let id = fun x -> x
 in id id
```

✓

```
let id x = x
 in id id
```

```
let f id = id id
 in f (fun x -> x)
```

```
(fun id -> id id)
  (fun x -> x)
```

Prenex quantifification

$\forall\alpha.\alpha \to \alpha$ ✓

$\forall\alpha.\forall\beta.\alpha \to (\beta \to \beta)$ ✓

$\forall\alpha.(\forall\beta.\beta \to \beta) \to \alpha$ ✗

$\forall\alpha.\alpha \to (\forall\beta.\beta \to \beta)$ ✗

Let-bound polymorphism

```
let id = fun x -> x
 in id id
```

✓

```
let id x = x
 in id id
```

✓

```
let f id = id id
 in f (fun x -> x)
```

```
(fun id -> id id)
  (fun x -> x)
```

Prenex quantifification

$\forall\alpha.\alpha \to \alpha$ ✓

$\forall\alpha.\forall\beta.\alpha \to (\beta \to \beta)$ ✓

$\forall\alpha.(\forall\beta.\beta \to \beta) \to \alpha$ ✗

$\forall\alpha.\alpha \to (\forall\beta.\beta \to \beta)$ ✗

Let-bound polymorphism

```
let id = fun x -> x
 in id id
```

✓

```
let id x = x
 in id id
```

✓

```
let f id = id id
 in f (fun x -> x)
```

✗

```
(fun id -> id id)
  (fun x -> x)
```

Prenex quantifification

$\forall\alpha.\alpha \to \alpha$ ✓

$\forall\alpha.\forall\beta.\alpha \to (\beta \to \beta)$ ✓

$\forall\alpha.(\forall\beta.\beta \to \beta) \to \alpha$ ✗

$\forall\alpha.\alpha \to (\forall\beta.\beta \to \beta)$ ✗

Let-bound polymorphism

```
let id = fun x -> x
 in id id
```

✓

```
let id x = x
 in id id
```

✓

```
let f id = id id
 in f (fun x -> x)
```

✗

```
(fun id -> id id)
  (fun x -> x)
```

✗

Types and schemes

$$\frac{}{\Gamma \vdash \mathcal{B} \text{ is a type}}\ \mathcal{B}\text{-types}$$

$$\frac{\alpha \in \Gamma}{\Gamma \vdash \alpha \text{ is a type}}\ \alpha\text{-types}$$

$$\frac{\Gamma \vdash A \text{ is a type} \qquad \Gamma \vdash B \text{ is a type}}{\Gamma \vdash A \to B \text{ is a type}}\ \to\text{-types}$$

$$\frac{\Gamma, \overline{\alpha} \vdash A \text{ is a type}}{\Gamma \vdash \forall \overline{\alpha}.A \text{ is a scheme}}\ \text{scheme}$$

Environments

$$\frac{}{\cdot \text{ is an environment}}\ \Gamma\text{-}\cdot$$

$$\frac{\Gamma \text{ is an environment} \qquad \Gamma \vdash S \text{ is a scheme}}{\Gamma, x : S \text{ is an environment}}\ \Gamma\text{-}:$$

Typing rules for $\to$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B} \to\text{-intro}$$

$$\frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B} \to\text{-elim}$$

Typing rules for schemes

$$\frac{\Gamma \vdash M : A \qquad \overline{\alpha} \cap ftv(\Gamma) = \emptyset \qquad \Gamma, x : \forall \overline{\alpha}.A \vdash N : B}{\Gamma \vdash \text{let x} = M \text{ in } N : B} \text{ scheme-intro}$$

$$\frac{\begin{array}{c} x : \forall \overline{\alpha}.A \in \Gamma \\ \Gamma \vdash B \text{ is a type} \qquad (\text{for } B \in \overline{B}) \end{array}}{\Gamma \vdash x : A[\overline{\alpha} := \overline{B}]} \text{ scheme-elim}$$

Milner’s algorithm

Substitutions

$$[a_1 \mapsto A_1, \; a_2 \mapsto A_2, \; \ldots \; a_n \mapsto A_n]$$

For example, let

σ be $[a \mapsto \mathcal{B}, \; b \mapsto (\mathcal{B} \to \mathcal{B})]$

A be $a \to b \to a$

Then

σA is $\mathcal{B} \to (\mathcal{B} \to \mathcal{B}) \to \mathcal{B}$.

If

$\sigma A = B$ (for some σ)

then we say

B is a *substitution instance* of A.

Constraints

$$a = b$$

$$a \rightarrow b = \mathcal{B} \rightarrow b$$

$$\mathcal{B} = \mathcal{B}$$

$$\mathcal{B} = \mathcal{B} \rightarrow \mathcal{B}$$

Unification

$$\text{unify} : \text{ConstraintSet} \to \text{Substitution}$$

$$\begin{aligned}
\text{unify}(\emptyset) &= [] \\
\text{unify}(\{A = A\} \cup C) &= \text{unify}(C) \\
\text{unify}(\{a = A\} \cup C) &= \text{unify}([a \mapsto A]C) \circ [a \mapsto A] \\
&\quad \text{when } a \notin \mathit{ftv}(A) \\
\text{unify}(\{A = a\} \cup C) &= \text{unify}([a \mapsto A]C) \circ [a \mapsto A] \\
&\quad \text{when } a \notin \mathit{ftv}(A) \\
\text{unify}(\{A \to B = A' \to B'\} \cup C) &= \text{unify}(\{A = A', B = B'\} \cup C) \\
\text{unify}(\{A = B\} \cup C) &= \mathit{FAIL}
\end{aligned}$$

Algorithm J

```
J : Environment × Expression → Type
```

J (Γ, λx.M) = $b \to A$
where A = J (Γ,x:b, M)
and b *is fresh*

J (Γ, M N) = b
where A = J (Γ, M)
and B = J (Γ, N)
and unify' ({A = $B \to b$})
succeeds
and b *is fresh*

J (Γ, x) = A[$\overline{\alpha}$:=$\overline{b}$]
where Γ(x) = $\forall\overline{\alpha}.A$
and $\overline{b}$ *are fresh*

J (Γ, let x = M in N) = B
where A = J (Γ, M)
and B = J (Γ,x:$\forall\overline{\alpha}.A$, N)
and $\overline{\alpha}$ = ftv(A) \ ftv(Γ)

Algorithm J in action

```
J(·, let apply = λf.λx.f x in
      let id = λy.y in
       apply id) =
```

Algorithm J in action

```
J(·, let apply = λf.λx.f x in
      let id = λy.y in
        apply id) =
  J(·, λf.λx.f x) =
```

Algorithm J in action

```
J(·, let apply = λf.λx.f x in
      let id = λy.y in
       apply id) =
  J(·, λf.λx.f x) =
    J(·,f:b₁, λx.f x) =
```

Algorithm J in action

$$
\begin{array}{l}
\texttt{J}(\cdot,\ \textbf{let}\ \texttt{apply} = \lambda\texttt{f}.\lambda\texttt{x}.\texttt{f x}\ \textbf{in} \\
\quad\quad \textbf{let}\ \texttt{id} = \lambda\texttt{y}.\texttt{y}\ \textbf{in} \\
\quad\quad\quad \texttt{apply id}) = \\
\quad \texttt{J}(\cdot,\ \lambda\texttt{f}.\lambda\texttt{x}.\texttt{f x}) = b_1 \to b_2 \to b_3 \\
\quad\quad \texttt{J}(\cdot,\texttt{f}:b_1,\ \lambda\texttt{x}.\texttt{f x}) = b_2 \to b_3 \\
\quad\quad\quad \texttt{J}(\cdot,\texttt{f}:b_1,\texttt{x}:b_2,\ \texttt{f x}) = b_3
\end{array}
$$

Algorithm J in action

```
J(·, let apply = λf.λx.f x in
      let id = λy.y in
       apply id) =
   J(·, λf.λx.f x) = b1 → b2 → b3
      J(·,f:b1, λx.f x) = b2 → b3
         J(·,f:b1,x:b2, f x) = b3
            J(·,f:b1,x:b2, f) =
```

Algorithm J in action

```
J(·, let apply = λf.λx.f x in
      let id = λy.y in
        apply id) =
  J(·, λf.λx.f x) = b1 → b2 → b3
    J(·,f:b1, λx.f x) = b2 → b3
      J(·,f:b1,x:b2, f x) = b3
        J(·,f:b1,x:b2, f) = b1
```

Algorithm J in action

```
J(·, let apply = λf.λx.f x in
      let id = λy.y in
       apply id) =
  J(·, λf.λx.f x) = b1 → b2 → b3
    J(·,f:b1, λx.f x) = b2 → b3
      J(·,f:b1,x:b2, f x) = b3
        J(·,f:b1,x:b2, f) = b1
        J(·,f:b1,x:b2, x) =
```

Algorithm J in action

```
J(·, let apply = λf.λx.f x in
     let id = λy.y in
      apply id) =
  J(·, λf.λx.f x) = b₁ → b₂ → b₃
    J(·,f:b₁, λx.f x) = b₂ → b₃
      J(·,f:b₁,x:b₂, f x) = b₃
        J(·,f:b₁,x:b₂, f) = b₁
        J(·,f:b₁,x:b₂, x) = b₂
```

Algorithm J in action

$$
\begin{array}{l}
\texttt{J}(\cdot,\ \textbf{let}\ \texttt{apply} = \lambda\texttt{f}.\lambda\texttt{x}.\texttt{f x}\ \textbf{in} \\
\quad\ \textbf{let}\ \texttt{id} = \lambda\texttt{y}.\texttt{y}\ \textbf{in} \\
\quad\quad \texttt{apply id}) = \\
\quad \texttt{J}(\cdot,\ \lambda\texttt{f}.\lambda\texttt{x}.\texttt{f x}) = b_1 \to b_2 \to b_3 \\
\quad\quad \texttt{J}(\cdot,\texttt{f}:b_1,\ \lambda\texttt{x}.\texttt{f x}) = b_2 \to b_3 \\
\quad\quad\quad \texttt{J}(\cdot,\texttt{f}:b_1,\texttt{x}:b_2,\ \texttt{f x}) = b_3 \\
\quad\quad\quad\quad \texttt{J}(\cdot,\texttt{f}:b_1,\texttt{x}:b_2,\ \texttt{f}) = b_1 \\
\quad\quad\quad\quad \texttt{J}(\cdot,\texttt{f}:b_1,\texttt{x}:b_2,\ \texttt{x}) = b_2 \\
\quad\quad\quad\quad \texttt{unify}(\{b_1 = b_2 \to b_3\}) =
\end{array}
$$

Algorithm J in action

$$
\begin{array}{l}
\texttt{J}(\cdot,\ \textbf{let}\ \texttt{apply} = \lambda\texttt{f}.\lambda\texttt{x}.\texttt{f}\ \texttt{x}\ \textbf{in}\\
\quad\ \ \textbf{let}\ \texttt{id} = \lambda\texttt{y}.\texttt{y}\ \textbf{in}\\
\quad\quad \texttt{apply}\ \texttt{id}) =\\
\quad \texttt{J}(\cdot,\ \lambda\texttt{f}.\lambda\texttt{x}.\texttt{f}\ \texttt{x}) = b_1 \to b_2 \to b_3\\
\quad\quad \texttt{J}(\cdot,\texttt{f}:b_1,\ \lambda\texttt{x}.\texttt{f}\ \texttt{x}) = b_2 \to b_3\\
\quad\quad\quad \texttt{J}(\cdot,\texttt{f}:b_1,\texttt{x}:b_2,\ \texttt{f}\ \texttt{x}) = b_3\\
\quad\quad\quad\quad \texttt{J}(\cdot,\texttt{f}:b_1,\texttt{x}:b_2,\ \texttt{f}) = b_1\\
\quad\quad\quad\quad \texttt{J}(\cdot,\texttt{f}:b_1,\texttt{x}:b_2,\ \texttt{x}) = b_2\\
\quad\quad\quad\quad \texttt{unify}(\{b_1 = b_2 \to b_3\}) = \{b_1 \mapsto b_2 \to b_3\}
\end{array}
$$

Algorithm J in action

$$
\begin{array}{l}
\texttt{J}(\cdot,\ \textbf{let}\ \texttt{apply} = \lambda\texttt{f}.\lambda\texttt{x}.\texttt{f x}\ \textbf{in} \\
\qquad \textbf{let}\ \texttt{id} = \lambda\texttt{y}.\texttt{y}\ \textbf{in} \\
\qquad\quad \texttt{apply id}) = \\
\quad \texttt{J}(\cdot,\ \lambda\texttt{f}.\lambda\texttt{x}.\texttt{f x}) = (b_2 \to b_3) \to b_2 \to b_3 \\
\qquad \texttt{J}(\cdot, \texttt{f}:b_2 \to b_3,\ \lambda\texttt{x}.\texttt{f x}) = b_2 \to b_3 \\
\qquad\quad \texttt{J}(\cdot, \texttt{f}:b_2 \to b_3, \texttt{x}:b_2,\ \texttt{f x}) = b_3 \\
\qquad\qquad \texttt{J}(\cdot, \texttt{f}:b_2 \to b_3, \texttt{x}:b_2,\ \texttt{f}) = b_2 \to b_3 \\
\qquad\qquad \texttt{J}(\cdot, \texttt{f}:b_2 \to b_3, \texttt{x}:b_2,\ \texttt{x}) = b_2 \\
\qquad\qquad \texttt{unify}(\{b_1 = b_2 \to b_3\}) = \{b_1 \mapsto b_2 \to b_3\}
\end{array}
$$

Algorithm J in action

$$
\begin{array}{l}
\texttt{J}(\cdot,\ \textbf{let}\ \texttt{apply} = \lambda\texttt{f}.\lambda\texttt{x}.\texttt{f}\ \texttt{x}\ \textbf{in} \\
\quad\quad \textbf{let}\ \texttt{id} = \lambda\texttt{y}.\texttt{y}\ \textbf{in} \\
\quad\quad\quad \texttt{apply}\ \texttt{id}) = \\
\quad \texttt{J}(\cdot,\ \lambda\texttt{f}.\lambda\texttt{x}.\texttt{f}\ \texttt{x}) = (b_2 \to b_3) \to b_2 \to b_3 \\
\quad\quad \texttt{J}(\cdot, \texttt{f}: b_2 \to b_3,\ \lambda\texttt{x}.\texttt{f}\ \texttt{x}) = b_2 \to b_3 \\
\quad\quad\quad \texttt{J}(\cdot, \texttt{f}: b_2 \to b_3, \texttt{x}: b_2,\ \texttt{f}\ \texttt{x}) = b_3 \\
\quad\quad\quad\quad \texttt{J}(\cdot, \texttt{f}: b_2 \to b_3, \texttt{x}: b_2,\ \texttt{f}) = b_2 \to b_3 \\
\quad\quad\quad\quad \texttt{J}(\cdot, \texttt{f}: b_2 \to b_3, \texttt{x}: b_2,\ \texttt{x}) = b_2 \\
\quad \texttt{ftv}((b_2 \to b_3) \to b_2 \to b_3) = \{b_2,\ b_3\} \\
\quad \texttt{ftv}(\cdot) = \{\} \\
\quad \{b_2,\ b_3\} \setminus \{\} = \{b_2,\ b_3\}
\end{array}
$$

Algorithm J in action

```
J(·, let apply = λf.λx.f x in
      let id = λy.y in
       apply id) =
   J(·, λf.λx.f x) = (b₂ → b₃) → b₂ → b₃
   J(·,apply:∀α₂α₃.(α₂ → α₃)→ α₂ → α₃,
     let id = λy.y in apply id) =
```

Algorithm J in action

$$
\begin{array}{l}
\texttt{J}(\cdot,\ \textbf{let}\ \texttt{apply} = \lambda\texttt{f}.\lambda\texttt{x}.\texttt{f}\ \texttt{x}\ \textbf{in} \\
\quad\ \ \textbf{let}\ \texttt{id} = \lambda\texttt{y}.\texttt{y}\ \textbf{in} \\
\quad\quad \texttt{apply}\ \texttt{id}) = \\
\quad \texttt{J}(\cdot,\ \lambda\texttt{f}.\lambda\texttt{x}.\texttt{f}\ \texttt{x}) = (b_2 \to b_3) \to b_2 \to b_3 \\
\quad \texttt{J}(\cdot,\texttt{apply}:\forall\alpha_2\alpha_3.(\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3, \\
\quad\quad \textbf{let}\ \texttt{id} = \lambda\texttt{y}.\texttt{y}\ \textbf{in}\ \texttt{apply}\ \texttt{id}) = \\
\quad\quad \texttt{J}(\cdot,\texttt{apply}:\forall\alpha_2\alpha_3.(\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3, \\
\quad\quad\quad \lambda\texttt{y}.\texttt{y}) =
\end{array}
$$

Algorithm J in action

```
J(·, let apply = λf.λx.f x in
      let id = λy.y in
       apply id) =
  J(·, λf.λx.f x) = (b2 → b3) → b2 → b3
  J(·,apply:∀α2α3.(α2 → α3)→ α2 → α3,
    let id = λy.y in apply id) =
    J(·,apply:∀α2α3.(α2 → α3)→ α2 → α3,
      λy.y) = b4 → b4
      J(·,apply:∀α2α3.(α2 → α3)→ α2 → α3,y:b4, y)
        = b4
```

Algorithm J in action

$$
\begin{array}{l}
\texttt{J}(\cdot,\ \textbf{let}\ \texttt{apply} = \lambda\texttt{f}.\lambda\texttt{x}.\texttt{f}\ \texttt{x}\ \textbf{in} \\
\quad\quad \textbf{let}\ \texttt{id} = \lambda\texttt{y}.\texttt{y}\ \textbf{in} \\
\quad\quad\quad \texttt{apply}\ \texttt{id}) = \\
\quad \texttt{J}(\cdot,\ \lambda\texttt{f}.\lambda\texttt{x}.\texttt{f}\ \texttt{x}) = (b_2 \to b_3) \to b_2 \to b_3 \\
\quad \texttt{J}(\cdot, \texttt{apply}: \forall\alpha_2\alpha_3.(\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3, \\
\quad\quad \textbf{let}\ \texttt{id} = \lambda\texttt{y}.\texttt{y}\ \textbf{in}\ \texttt{apply}\ \texttt{id}) = \\
\quad\quad \texttt{J}(\cdot, \texttt{apply}: \forall\alpha_2\alpha_3.(\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3, \\
\quad\quad\quad \lambda\texttt{y}.\texttt{y}) = b_4 \to b_4 \\
\quad\quad \texttt{ftv}(b_4 \to b_4) = \{b_4\} \\
\quad\quad \texttt{ftv}(\cdot, \texttt{apply}: \forall\alpha_2\alpha_3.(\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3) = \{\} \\
\quad\quad \{b_4\} \setminus \{\} = \{b_4\}
\end{array}
$$

Algorithm J in action

```
J(·, let apply = λf.λx.f x in
      let id = λy.y in
       apply id) =
   J(·, λf.λx.f x) = (b2 → b3) → b2 → b3
   J(·,apply:∀α2α3.(α2 → α3)→ α2 → α3,
     let id = λy.y in apply id) =
     J(·,apply:∀α2α3.(α2 → α3)→ α2 → α3,
       λy.y) = b4 → b4
     J(·, apply:∀α2α3.(α2 → α3) → α2 → α3,id:∀α4.α4 → α4,
       apply id) = b5
```

Algorithm J in action

$$
\begin{array}{l}
\texttt{J}(\cdot,\ \textbf{let}\ \texttt{apply = }\lambda\texttt{f.}\lambda\texttt{x.f x}\ \textbf{in} \\
\quad\quad \textbf{let}\ \texttt{id = }\lambda\texttt{y.y}\ \textbf{in} \\
\quad\quad\quad \texttt{apply id) =} \\
\quad \texttt{J}(\cdot,\ \lambda\texttt{f.}\lambda\texttt{x.f x) = }(b_2 \to b_3) \to b_2 \to b_3 \\
\quad \texttt{J}(\cdot,\texttt{apply}:\forall\alpha_2\alpha_3.(\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3, \\
\quad\quad \textbf{let}\ \texttt{id = }\lambda\texttt{y.y}\ \textbf{in}\ \texttt{apply id) =} \\
\quad\quad \texttt{J}(\cdot,\texttt{apply}:\forall\alpha_2\alpha_3.(\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3, \\
\quad\quad\quad \lambda\texttt{y.y) = }b_4 \to b_4 \\
\quad\quad \texttt{J}(\cdot,\ \texttt{apply}:\forall\alpha_2\alpha_3.(\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3, \texttt{id}:\forall\alpha_4.\alpha_4 \to \alpha_4, \\
\quad\quad\quad \texttt{apply id) = }b_5 \\
\quad\quad\quad \texttt{J}(\cdot,\ \texttt{apply}:\forall\alpha_2\alpha_3.(\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3, \\
\quad\quad\quad\quad\quad \texttt{id}:\forall\alpha_4.\alpha_4 \to \alpha_4,\ \texttt{apply)} \\
\quad\quad\quad\quad \texttt{= }(b_6 \to b_7) \to b_6 \to b_7
\end{array}
$$

Algorithm J in action

```
J(·, let apply = λf.λx.f x in
      let id = λy.y in
       apply id) =
   J(·, λf.λx.f x) = (b2 → b3) → b2 → b3
   J(·,apply:∀α2α3.(α2 → α3)→ α2 → α3,
     let id = λy.y in apply id) =
     J(·,apply:∀α2α3.(α2 → α3)→ α2 → α3,
       λy.y) = b4 → b4
     J(·, apply:∀α2α3.(α2 → α3) → α2 → α3,id:∀α4.α4 → α4,
       apply id) = b5
       J(·, apply:∀α2α3.(α2 → α3) → α2 → α3,
            id:∀α4.α4 → α4, apply)
          = (b6 → b7)→ b6 → b7
       J(·, apply:∀α2α3.(α2 → α3) → α2 → α3,
            id:∀α4.α4 → α4, id)
          = b8 → b8
```

Algorithm J in action

$$\texttt{unify}\ (\{(b_6 \to b_7) \to b_6 \to b_7\ \texttt{=}\ (b_8 \to b_8)\ \to\ b_5\})$$

Algorithm J in action

$$
\begin{aligned}
& \texttt{unify}\ (\{(b_6 \to b_7) \to b_6 \to b_7\ \texttt{=}\ (b_8 \to b_8)\ \to\ b_5\}) \\
\texttt{=}\ & \texttt{unify}\ (\{b_6 \to b_7\ \texttt{=}\ b_8 \to b_8\texttt{,} \\
& \qquad\qquad b_6 \to b_7\ \texttt{=}\ b_5\})
\end{aligned}
$$

Algorithm J in action

$$
\begin{aligned}
& \texttt{unify}\ (\{(b_6 \to b_7) \to b_6 \to b_7 \ \texttt{=}\ (b_8 \to b_8) \ \to\ b_5\}) \\
= \ & \texttt{unify}\ (\{b_6 \to b_7 \ \texttt{=}\ b_8 \to b_8\,\texttt{,} \\
& \qquad\quad b_6 \to b_7 \ \texttt{=}\ b_5\}) \\
= \ & \texttt{unify}\ (\{b_6 \ \texttt{=}\ b_8\,\texttt{,} \\
& \qquad\quad b_7 \ \texttt{=}\ b_8\,\texttt{,} \\
& \qquad\quad b_6 \to b_7 \ \texttt{=}\ b_5\})
\end{aligned}
$$

Algorithm J in action

$$
\begin{aligned}
& \texttt{unify}\ (\{(b_6 \to b_7) \to b_6 \to b_7\ \texttt{=}\ (b_8 \to b_8) \to b_5\}) \\
\texttt{=}\ & \texttt{unify}\ (\{b_6 \to b_7\ \texttt{=}\ b_8 \to b_8\texttt{,} \\
& \qquad\qquad b_6 \to b_7\ \texttt{=}\ b_5\}) \\
\texttt{=}\ & \texttt{unify}\ (\{b_6\ \texttt{=}\ b_8\texttt{,} \\
& \qquad\qquad b_7\ \texttt{=}\ b_8\texttt{,} \\
& \qquad\qquad b_6 \to b_7\ \texttt{=}\ b_5\}) \\
\texttt{=}\ & \{b_6 \mapsto b_8\texttt{,}\ b_7 \mapsto b_8\texttt{,}\ b_5 \mapsto b_6 \to b_7\}
\end{aligned}
$$

Algorithm J in action

$$
\begin{array}{l}
\mathtt{J}(\cdot,\ \mathbf{let}\ \mathtt{apply} = \lambda \mathtt{f}.\lambda \mathtt{x}.\mathtt{f}\ \mathtt{x}\ \mathbf{in} \\
\quad\quad \mathbf{let}\ \mathtt{id} = \lambda \mathtt{y}.\mathtt{y}\ \mathbf{in} \\
\quad\quad\quad \mathtt{apply}\ \mathtt{id}) = \\
\quad \mathtt{J}(\cdot,\ \lambda \mathtt{f}.\lambda \mathtt{x}.\mathtt{f}\ \mathtt{x}) = (b_2 \to b_3) \to b_2 \to b_3 \\
\quad \mathtt{J}(\cdot, \mathtt{apply}: \forall \alpha_2 \alpha_3.(\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3, \\
\quad\quad \mathbf{let}\ \mathtt{id} = \lambda \mathtt{y}.\mathtt{y}\ \mathbf{in}\ \mathtt{apply}\ \mathtt{id}) = \\
\quad\quad \mathtt{J}(\cdot, \mathtt{apply}: \forall \alpha_2 \alpha_3.(\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3, \\
\quad\quad\quad \lambda \mathtt{y}.\mathtt{y}) = b_4 \to b_4 \\
\quad\quad \mathtt{J}(\cdot,\ \mathtt{apply}: \forall \alpha_2 \alpha_3.(\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3, \mathtt{id}: \forall \alpha_4.\alpha_4 \to \alpha_4, \\
\quad\quad\quad \mathtt{apply}\ \mathtt{id}) = b_5 \\
\quad\quad\quad \mathtt{J}(\cdot,\ \mathtt{apply}: \forall \alpha_2 \alpha_3.(\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3, \\
\quad\quad\quad\quad\quad \mathtt{id}: \forall \alpha_4.\alpha_4 \to \alpha_4,\ \mathtt{apply}) \\
\quad\quad\quad\quad = (b_6 \to b_7) \to b_6 \to b_7 \\
\quad\quad\quad \mathtt{J}(\cdot,\ \mathtt{apply}: \forall \alpha_2 \alpha_3.(\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3, \\
\quad\quad\quad\quad\quad \mathtt{id}: \forall \alpha_4.\alpha_4 \to \alpha_4,\ \mathtt{id}) \\
\quad\quad\quad\quad = b_8 \to b_8
\end{array}
$$

Algorithm J in action

$$
\begin{array}{l}
\texttt{J}(\cdot,\ \textbf{let}\ \texttt{apply} = \lambda\texttt{f}.\lambda\texttt{x}.\texttt{f x}\ \textbf{in} \\
\qquad \textbf{let}\ \texttt{id} = \lambda\texttt{y}.\texttt{y}\ \textbf{in} \\
\qquad\quad \texttt{apply id}) = \\
\quad \texttt{J}(\cdot,\ \lambda\texttt{f}.\lambda\texttt{x}.\texttt{f x}) = (b_2 \to b_3) \to b_2 \to b_3 \\
\quad \texttt{J}(\cdot, \texttt{apply}:\forall\alpha_2\alpha_3.(\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3, \\
\qquad \textbf{let}\ \texttt{id} = \lambda\texttt{y}.\texttt{y}\ \textbf{in}\ \texttt{apply id}) = \\
\qquad \texttt{J}(\cdot, \texttt{apply}:\forall\alpha_2\alpha_3.(\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3, \\
\qquad\quad \lambda\texttt{y}.\texttt{y}) = b_4 \to b_4 \\
\qquad \texttt{J}(\cdot,\ \texttt{apply}:\forall\alpha_2\alpha_3.(\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3, \texttt{id}:\forall\alpha_4.\alpha_4 \to \alpha_4, \\
\qquad\quad \texttt{apply id}) = b_8 \to b_8 \\
\qquad\quad \texttt{J}(\cdot,\ \texttt{apply}:\forall\alpha_2\alpha_3.(\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3, \\
\qquad\qquad\quad \texttt{id}:\forall\alpha_4.\alpha_4 \to \alpha_4,\ \texttt{apply}) \\
\qquad\qquad = (b_8 \to b_8) \to b_8 \to b_8 \\
\qquad\quad \texttt{J}(\cdot,\ \texttt{apply}:\forall\alpha_2\alpha_3.(\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3, \\
\qquad\qquad\quad \texttt{id}:\forall\alpha_4.\alpha_4 \to \alpha_4,\ \texttt{id}) \\
\qquad\qquad = b_8 \to b_8
\end{array}
$$

Algorithm J in action

$$
\begin{array}{l}
\texttt{J}(\cdot,\ \textbf{let}\ \texttt{apply = }\lambda\texttt{f.}\lambda\texttt{x.f x}\ \textbf{in} \\
\quad\quad \textbf{let}\ \texttt{id = }\lambda\texttt{y.y}\ \textbf{in} \\
\quad\quad\quad \texttt{apply id) =} \\
\quad \texttt{J}(\cdot,\ \lambda\texttt{f.}\lambda\texttt{x.f x) = } (b_2 \to b_3) \to b_2 \to b_3 \\
\quad \texttt{J}(\cdot,\texttt{apply}:\forall\alpha_2\alpha_3.(\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3, \\
\quad\quad \textbf{let}\ \texttt{id = }\lambda\texttt{y.y}\ \textbf{in}\ \texttt{apply id) =} \\
\quad\quad \texttt{J}(\cdot,\texttt{apply}:\forall\alpha_2\alpha_3.(\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3, \\
\quad\quad\quad \lambda\texttt{y.y) = } b_4 \to b_4 \\
\quad\quad \texttt{J}(\cdot,\ \texttt{apply}:\forall\alpha_2\alpha_3.(\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3, \texttt{id}:\forall\alpha_4.\alpha_4 \to \alpha_4, \\
\quad\quad\quad \texttt{apply id) = } b_8 \to b_8
\end{array}
$$

Algorithm J in action

$$
\begin{array}{l}
\texttt{J}(\cdot,\ \textbf{let}\ \texttt{apply} = \lambda\texttt{f}.\lambda\texttt{x}.\texttt{f x}\ \textbf{in} \\
\quad\quad \textbf{let}\ \texttt{id} = \lambda\texttt{y}.\texttt{y}\ \textbf{in} \\
\quad\quad\quad \texttt{apply id}) = \\
\quad \texttt{J}(\cdot,\ \lambda\texttt{f}.\lambda\texttt{x}.\texttt{f x}) = (b_2 \to b_3) \to b_2 \to b_3 \\
\quad \texttt{J}(\cdot, \texttt{apply}:\forall\alpha_2\alpha_3.(\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3, \\
\quad\quad \textbf{let}\ \texttt{id} = \lambda\texttt{y}.\texttt{y}\ \textbf{in}\ \texttt{apply id}) = b_8 \to b_8 \\
\quad\quad \texttt{J}(\cdot,\ \texttt{apply}:\forall\alpha_2\alpha_3.(\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3, \texttt{id}:\forall\alpha_4.\alpha_4 \to \alpha_4, \\
\quad\quad\quad \texttt{apply id}) = b_8 \to b_8
\end{array}
$$

Algorithm J in action

```
J(·, let apply = λf.λx.f x in
      let id = λy.y in
       apply id) = b8 → b8
```

Type inference in practice

Type inference and recursion

$$\frac{\Gamma, x : A \vdash M : A \qquad \overline{\alpha} \notin ftv(\Gamma) \qquad \Gamma, x : \forall\overline{\alpha}.A \vdash N : B}{\Gamma \vdash \text{let rec x = M in N} : B} \text{let-rec}$$

Supporting imperative programming: the value restriction

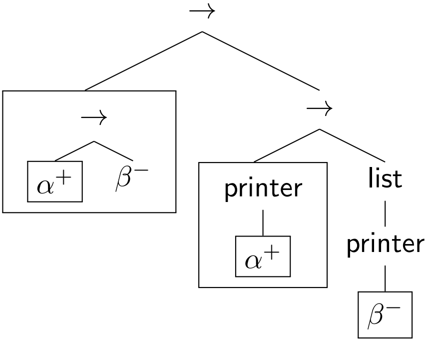
```
type 'a ref = { mutable contents : 'a }
val ref : 'a -> 'a ref
val ( ! ) : 'a ref -> 'a
val (:=) : 'a ref -> 'a -> unit

let r = ref None in
   r := Some "boom";
   match !r with
     None -> ()
   | Some f -> f ()
```

Relaxing the value restriction: variance

```
type 'a printer = 'a -> string

('a -> 'b) -> 'a printer -> 'b printer list
```



Relaxing the value restriction: the rules

Should we generalize?

- covariant type variables
- invariant type variables
- contravariant type variables
- bivariant type variables

Relaxing the value restriction: the rules

Should we generalize?

- covariant type variables ✓
- invariant type variables
- contravariant type variables
- bivariant type variables

Relaxing the value restriction: the rules

Should we generalize?

- covariant type variables ✓
- invariant type variables ✗
- contravariant type variables
- bivariant type variables

Relaxing the value restriction: the rules

Should we generalize?

- covariant type variables ✓
- invariant type variables ✗
- contravariant type variables ✗
- bivariant type variables

Relaxing the value restriction: the rules

Should we generalize?

- covariant type variables ✓
- invariant type variables ✗
- contravariant type variables ✗
- bivariant type variables ✓

Next time

$$\frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B}$$

$$\frac{\Gamma \vdash A \to B \qquad \Gamma \vdash A}{\Gamma \vdash B}$$