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

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

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

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

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

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

Environments

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

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

$$\frac{\Gamma \text{ is an environment}}{\Gamma, \alpha \text{ is an environment}}\ \Gamma\text{-}::$$

Typing rules for $\rightarrow$

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

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

Typing rules for schemes

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

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

Milner's algorithm

Substitutions

$$\{\alpha_1 \;\mapsto\; \mathsf{A}_1\,,\;\; \alpha_2 \;\mapsto\; \mathsf{A}_2\,,\;\; \ldots\;\; \alpha_n \;\mapsto\; \mathsf{A}_n\}$$

For example, let

σ be $\{\alpha \mapsto \mathcal{B}, \beta \mapsto (\mathcal{B} \to \mathcal{B})\}$

A be $\alpha \to \beta \to \alpha$

Then

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

If

$\sigma \mathsf{A} = \mathsf{B}$ (for some σ)

then we say

B is a *substitution instance* of A.

Constraints

$$\alpha = \beta$$

$$\alpha \to \beta = \mathcal{B} \to \beta$$

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

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

Unification

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

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

Algorithm J

$$\text{J} \ : \ \text{Environment} \times \text{Expression} \ \rightarrow \ \text{Type}$$

$\text{J} \ (\Gamma, \ \lambda x.\text{M}) = \beta \rightarrow \text{A}$
where $\text{A} = \text{J} \ (\Gamma, x:\beta, \ \text{M})$
and β *is fresh*

$\text{J} \ (\Gamma, \ \text{M N}) = \beta$
where $\text{A} = \text{J} \ (\Gamma, \ \text{M})$
and $\text{B} = \text{J} \ (\Gamma, \ \text{N})$
and $\text{unify'} \ (\{\text{A} = \text{B} \rightarrow \beta\})$
succeeds
and β *is fresh*

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

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

Algorithm J in action

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

Algorithm J in action

$$
\begin{aligned}
J(\cdot,\ & \textbf{let}\ \mathsf{apply} = \lambda f.\lambda x.f\ x\ \textbf{in} \\
& \quad \textbf{let}\ \mathsf{id} = \lambda y.y\ \textbf{in} \\
& \quad\quad \mathsf{apply}\ \mathsf{id}) = \\
& J(\cdot,\ \lambda f.\lambda x.f\ x) =
\end{aligned}
$$

Algorithm J in action

$$
\begin{array}{l}
J(\cdot,\ \textbf{let}\ apply = \lambda f.\lambda x.f\ x\ \textbf{in} \\
\quad \textbf{let}\ id = \lambda y.y\ \textbf{in} \\
\quad\quad apply\ id) = \\
\quad J(\cdot,\ \lambda f.\lambda x.f\ x) = \\
\quad\quad J(\cdot, f:\beta_1,\ \lambda x.f\ x) =
\end{array}
$$

Algorithm J in action

```
J(·, let apply = λf.λx.f x in
       let id = λy.y in
         apply id) =
   J(·, λf.λx.f x) = β1 → β2 → β3
      J(·,f:β1, λx.f x) = β2 → β3
         J(·,f:β1,x:β2, f x) = β3
```

Algorithm J in action

$$
\begin{array}{l}
\mathsf{J}(\cdot,\ \textbf{let}\ \mathsf{apply} = \lambda \mathsf{f}.\lambda \mathsf{x}.\mathsf{f}\ \mathsf{x}\ \textbf{in} \\
\quad\ \textbf{let}\ \mathsf{id} = \lambda \mathsf{y}.\mathsf{y}\ \textbf{in} \\
\quad\quad \mathsf{apply}\ \mathsf{id}) = \\
\quad \mathsf{J}(\cdot,\ \lambda \mathsf{f}.\lambda \mathsf{x}.\mathsf{f}\ \mathsf{x}) = \beta_1 \to \beta_2 \to \beta_3 \\
\quad\quad \mathsf{J}(\cdot, \mathsf{f}:\beta_1,\ \lambda \mathsf{x}.\mathsf{f}\ \mathsf{x}) = \beta_2 \to \beta_3 \\
\quad\quad\quad \mathsf{J}(\cdot, \mathsf{f}:\beta_1, \mathsf{x}:\beta_2,\ \mathsf{f}\ \mathsf{x}) = \beta_3 \\
\quad\quad\quad\quad \mathsf{J}(\cdot, \mathsf{f}:\beta_1, \mathsf{x}:\beta_2,\ \mathsf{f}) =
\end{array}
$$

Algorithm J in action

```
J(·, let apply = λf.λx.f x in
       let id = λy.y in
         apply id) =
   J(·, λf.λx.f x) = β1 → β2 → β3
     J(·,f:β1, λx.f x) = β2 → β3
       J(·,f:β1,x:β2, f x) = β3
         J(·,f:β1,x:β2, f) = β1
```

Algorithm J in action

$$\begin{aligned}
&J(\cdot,\ \textbf{let}\ \text{apply} = \lambda f.\lambda x.f\ x\ \textbf{in}\\
&\quad\quad \textbf{let}\ \text{id} = \lambda y.y\ \textbf{in}\\
&\quad\quad\quad \text{apply id}) =\\
&\quad J(\cdot,\ \lambda f.\lambda x.f\ x) = \beta_1 \to \beta_2 \to \beta_3\\
&\quad\quad J(\cdot, f:\beta_1,\ \lambda x.f\ x) = \beta_2 \to \beta_3\\
&\quad\quad\quad J(\cdot, f:\beta_1, x:\beta_2,\ f\ x) = \beta_3\\
&\quad\quad\quad\quad J(\cdot, f:\beta_1, x:\beta_2,\ f) = \beta_1\\
&\quad\quad\quad\quad J(\cdot, f:\beta_1, x:\beta_2,\ x) =
\end{aligned}$$

Algorithm J in action

$$
\begin{array}{l}
J(\cdot,\ \textbf{let}\ \text{apply} = \lambda f.\lambda x.f\ x\ \textbf{in} \\
\quad \textbf{let}\ \text{id} = \lambda y.y\ \textbf{in} \\
\quad\quad \text{apply id}) = \\
\quad J(\cdot,\ \lambda f.\lambda x.f\ x) = \beta_1 \to \beta_2 \to \beta_3 \\
\quad\quad J(\cdot, f:\beta_1,\ \lambda x.f\ x) = \beta_2 \to \beta_3 \\
\quad\quad\quad J(\cdot, f:\beta_1, x:\beta_2,\ f\ x) = \beta_3 \\
\quad\quad\quad\quad J(\cdot, f:\beta_1, x:\beta_2,\ f) = \beta_1 \\
\quad\quad\quad\quad J(\cdot, f:\beta_1, x:\beta_2,\ x) = \beta_2
\end{array}
$$

Algorithm J in action

$$
\begin{aligned}
&J(\cdot,\ \textbf{let}\ \text{apply} = \lambda f.\lambda x.f\ x\ \textbf{in}\\
&\quad\ \ \textbf{let}\ \text{id} = \lambda y.y\ \textbf{in}\\
&\quad\ \ \ \text{apply id}) =\\
&\quad J(\cdot,\ \lambda f.\lambda x.f\ x) = \beta_1 \to \beta_2 \to \beta_3\\
&\quad\quad J(\cdot, f:\beta_1,\ \lambda x.f\ x) = \beta_2 \to \beta_3\\
&\quad\quad\quad J(\cdot, f:\beta_1, x:\beta_2,\ f\ x) = \beta_3\\
&\quad\quad\quad\quad J(\cdot, f:\beta_1, x:\beta_2,\ f) = \beta_1\\
&\quad\quad\quad\quad J(\cdot, f:\beta_1, x:\beta_2,\ x) = \beta_2\\
&\quad\quad\quad\quad \text{unify}(\{\beta_1 = \beta_2 \to \beta_3\}) =
\end{aligned}
$$

Algorithm J in action

$$
\begin{aligned}
&\mathsf{J}(\cdot,\ \textbf{let}\ \mathsf{apply} = \lambda \mathsf{f}.\lambda \mathsf{x}.\mathsf{f}\ \mathsf{x}\ \textbf{in}\\
&\quad \textbf{let}\ \mathsf{id} = \lambda \mathsf{y}.\mathsf{y}\ \textbf{in}\\
&\quad\quad \mathsf{apply}\ \mathsf{id}) =\\
&\quad \mathsf{J}(\cdot,\ \lambda \mathsf{f}.\lambda \mathsf{x}.\mathsf{f}\ \mathsf{x}) = \beta_1 \to \beta_2 \to \beta_3\\
&\quad\quad \mathsf{J}(\cdot, \mathsf{f}:\beta_1,\ \lambda \mathsf{x}.\mathsf{f}\ \mathsf{x}) = \beta_2 \to \beta_3\\
&\quad\quad\quad \mathsf{J}(\cdot, \mathsf{f}:\beta_1, \mathsf{x}:\beta_2,\ \mathsf{f}\ \mathsf{x}) = \beta_3\\
&\quad\quad\quad\quad \mathsf{J}(\cdot, \mathsf{f}:\beta_1, \mathsf{x}:\beta_2,\ \mathsf{f}) = \beta_1\\
&\quad\quad\quad\quad \mathsf{J}(\cdot, \mathsf{f}:\beta_1, \mathsf{x}:\beta_2,\ \mathsf{x}) = \beta_2\\
&\quad\quad\quad\quad \mathsf{unify}(\{\beta_1 = \beta_2 \to \beta_3\}) = \{\beta_1 \mapsto \beta_2 \to \beta_3\}
\end{aligned}
$$

Algorithm J in action

$$
\begin{array}{l}
J(\cdot,\ \textbf{let}\ apply = \lambda f.\lambda x.f\ x\ \textbf{in} \\
\quad \textbf{let}\ id = \lambda y.y\ \textbf{in} \\
\quad\quad apply\ id) = \\
\quad J(\cdot,\ \lambda f.\lambda x.f\ x) = (\beta_2 \to \beta_3) \to \beta_2 \to \beta_3 \\
\quad\quad J(\cdot, f:\beta_2 \to \beta_3,\ \lambda x.f\ x) = \beta_2 \to \beta_3 \\
\quad\quad\quad J(\cdot, f:\beta_2 \to \beta_3, x:\beta_2,\ f\ x) = \beta_3 \\
\quad\quad\quad\quad J(\cdot, f:\beta_2 \to \beta_3, x:\beta_2,\ f) = \beta_2 \to \beta_3 \\
\quad\quad\quad\quad J(\cdot, f:\beta_2 \to \beta_3, x:\beta_2,\ x) = \beta_2 \\
\quad\quad\quad\quad unify(\{\beta_1 = \beta_2 \to \beta_3\}) = \{\beta_1 \mapsto \beta_2 \to \beta_3\}
\end{array}
$$

Algorithm J in action

$$
\begin{array}{l}
J(\cdot,\ \textbf{let}\ \mathsf{apply} = \lambda f.\lambda x.f\ x\ \textbf{in} \\
\quad \textbf{let}\ \mathsf{id} = \lambda y.y\ \textbf{in} \\
\quad\quad \mathsf{apply}\ \mathsf{id}) = \\
\quad J(\cdot,\ \lambda f.\lambda x.f\ x) = (\beta_2 \to \beta_3) \to \beta_2 \to \beta_3 \\
\quad\quad J(\cdot, f:\beta_2 \to \beta_3,\ \lambda x.f\ x) = \beta_2 \to \beta_3 \\
\quad\quad\quad J(\cdot, f:\beta_2 \to \beta_3, x:\beta_2,\ f\ x) = \beta_3 \\
\quad\quad\quad\quad J(\cdot, f:\beta_2 \to \beta_3, x:\beta_2,\ f) = \beta_2 \to \beta_3 \\
\quad\quad\quad\quad J(\cdot, f:\beta_2 \to \beta_3, x:\beta_2,\ x) = \beta_2 \\
\quad \mathsf{ftv}((\beta_2 \to \beta_3) \to \beta_2 \to \beta_3) = \{\beta_2,\ \beta_3\} \\
\quad \mathsf{ftv}(\cdot) = \{\} \\
\quad \{\beta_2,\ \beta_3\} \setminus \{\} = \{\beta_2,\ \beta_3\}
\end{array}
$$

Algorithm J in action

```
J(·, let apply = λf.λx.f x in
      let id = λy.y in
        apply id) =
   J(·, λf.λx.f x) = (β2 → β3) → β2 → β3
   J(·,apply:∀β2β3.(β2 → β3)→ β2 → β3,
      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)  =  (β2 → β3) → β2 → β3
   J(·, apply:∀β2β3.(β2 → β3)→ β2 → β3 ,
      let  id  = λy.y  in  apply  id) =
      J(·, apply:∀β2β3.(β2 → β3)→ β2 → β3 ,
         λy.y) =
```

Algorithm J in action

$$
\begin{array}{l}
J(\cdot,\ \textbf{let}\ \text{apply} = \lambda f.\lambda x.f\ x\ \textbf{in} \\
\quad \textbf{let}\ \text{id} = \lambda y.y\ \textbf{in} \\
\quad\quad \text{apply id}) = \\
\quad J(\cdot,\ \lambda f.\lambda x.f\ x) = (\beta_2 \to \beta_3) \to \beta_2 \to \beta_3 \\
\quad J(\cdot, \text{apply} : \forall \beta_2 \beta_3 . (\beta_2 \to \beta_3) \to \beta_2 \to \beta_3, \\
\quad\quad \textbf{let}\ \text{id} = \lambda y.y\ \textbf{in}\ \text{apply id}) = \\
\quad\quad J(\cdot, \text{apply} : \forall \beta_2 \beta_3 . (\beta_2 \to \beta_3) \to \beta_2 \to \beta_3, \\
\quad\quad\quad \lambda y.y) = \beta_4 \to \beta_4 \\
\quad\quad\quad J(\cdot, \text{apply} : \forall \beta_2 \beta_3 . (\beta_2 \to \beta_3) \to \beta_2 \to \beta_3, y : \beta_4,\ y) \\
\quad\quad\quad\quad = \beta_4
\end{array}
$$

Algorithm J in action

$$J(\cdot,\ \textbf{let}\ apply = \lambda f.\lambda x.f\ x\ \textbf{in}$$
$$\textbf{let}\ id = \lambda y.y\ \textbf{in}$$
$$apply\ id) =$$
$$J(\cdot,\ \lambda f.\lambda x.f\ x) = (\beta_2 \to \beta_3) \to \beta_2 \to \beta_3$$
$$J(\cdot, apply : \forall \beta_2 \beta_3.(\beta_2 \to \beta_3) \to \beta_2 \to \beta_3,$$
$$\textbf{let}\ id = \lambda y.y\ \textbf{in}\ apply\ id) =$$
$$J(\cdot, apply : \forall \beta_2 \beta_3.(\beta_2 \to \beta_3) \to \beta_2 \to \beta_3,$$
$$\lambda y.y) = \beta_4 \to \beta_4$$
$$ftv(\beta_4 \to \beta_4) = \{\beta_4\}$$
$$ftv(\cdot, apply : \forall \beta_2 \beta_3.(\beta_2 \to \beta_3) \to \beta_2 \to \beta_3) = \{\}$$
$$\{\beta_4\} \setminus \{\} = \{\beta_4\}$$

Algorithm J in action

$$
\begin{array}{l}
J(\cdot,\ \textbf{let}\ apply = \lambda f.\lambda x.f\ x\ \textbf{in} \\
\quad \textbf{let}\ id = \lambda y.y\ \textbf{in} \\
\quad\quad apply\ id) = \\
\quad J(\cdot,\ \lambda f.\lambda x.f\ x) = (\beta_2 \to \beta_3) \to \beta_2 \to \beta_3 \\
\quad J(\cdot, apply : \forall \beta_2 \beta_3.(\beta_2 \to \beta_3) \to \beta_2 \to \beta_3, \\
\quad\quad \textbf{let}\ id = \lambda y.y\ \textbf{in}\ apply\ id) = \\
\quad\quad J(\cdot, apply : \forall \beta_2 \beta_3.(\beta_2 \to \beta_3) \to \beta_2 \to \beta_3, \\
\quad\quad\quad \lambda y.y) = \beta_4 \to \beta_4 \\
\quad\quad J(\cdot,\ apply : \forall \beta_2 \beta_3.(\beta_2 \to \beta_3) \to \beta_2 \to \beta_3, id : \forall \beta_4.\beta_4 \to \beta_4, \\
\quad\quad\quad apply\ id) = \beta_5
\end{array}
$$

Algorithm J in action

```
J(·, let apply = λf.λx.f x in
       let id = λy.y in
        apply id) =
  J(·, λf.λx.f x) = (β2 → β3) → β2 → β3
  J(·,apply:∀β2β3.(β2 → β3)→ β2 → β3,
     let id = λy.y in apply id) =
     J(·,apply:∀β2β3.(β2 → β3)→ β2 → β3,
       λy.y) = β4 → β4
     J(·, apply:∀β2β3.(β2 → β3) → β2 → β3, id:∀β4.β4 → β4,
       apply id) = β5
       J(·, apply:∀β2β3.(β2 → β3) → β2 → β3,
              id:∀β4.β4 → β4, apply)
          = (β6 → β7)→ β6 → β7
```

Algorithm J in action

$$
\begin{array}{l}
J(\cdot,\ \textbf{let}\ \mathsf{apply} = \lambda f.\lambda x.f\ x\ \textbf{in} \\
\quad \textbf{let}\ \mathsf{id} = \lambda y.y\ \textbf{in} \\
\quad\quad \mathsf{apply}\ \mathsf{id}) = \\
\quad J(\cdot,\ \lambda f.\lambda x.f\ x) = (\beta_2 \to \beta_3) \to \beta_2 \to \beta_3 \\
\quad J(\cdot, \mathsf{apply} : \forall \beta_2\beta_3.(\beta_2 \to \beta_3) \to \beta_2 \to \beta_3, \\
\quad\quad \textbf{let}\ \mathsf{id} = \lambda y.y\ \textbf{in}\ \mathsf{apply}\ \mathsf{id}) = \\
\quad\quad J(\cdot, \mathsf{apply} : \forall \beta_2\beta_3.(\beta_2 \to \beta_3) \to \beta_2 \to \beta_3, \\
\quad\quad\quad \lambda y.y) = \beta_4 \to \beta_4 \\
\quad\quad J(\cdot,\ \mathsf{apply} : \forall \beta_2\beta_3.(\beta_2 \to \beta_3) \to \beta_2 \to \beta_3, \mathsf{id} : \forall \beta_4.\beta_4 \to \beta_4, \\
\quad\quad\quad \mathsf{apply}\ \mathsf{id}) = \beta_5 \\
\quad\quad\quad J(\cdot,\ \mathsf{apply} : \forall \beta_2\beta_3.(\beta_2 \to \beta_3) \to \beta_2 \to \beta_3, \\
\quad\quad\quad\quad\quad \mathsf{id} : \forall \beta_4.\beta_4 \to \beta_4,\ \mathsf{apply}) \\
\quad\quad\quad\quad = (\beta_6 \to \beta_7) \to \beta_6 \to \beta_7 \\
\quad\quad\quad J(\cdot,\ \mathsf{apply} : \forall \beta_2\beta_3.(\beta_2 \to \beta_3) \to \beta_2 \to \beta_3, \\
\quad\quad\quad\quad\quad \mathsf{id} : \forall \beta_4.\beta_4 \to \beta_4,\ \mathsf{id}) \\
\quad\quad\quad\quad = \beta_8 \to \beta_8
\end{array}
$$

Algorithm J in action

$$\texttt{unify}\ \ (\{(\beta_6 \to \beta_7) \to \beta_6 \to \beta_7 \ = \ (\beta_8 \to \beta_8) \ \to \ \beta_5\})$$

Algorithm J in action

$$
\begin{aligned}
& \texttt{unify}\ (\{(\beta_6 \rightarrow \beta_7) \rightarrow \beta_6 \rightarrow \beta_7 \ = \ (\beta_8 \rightarrow \beta_8) \ \rightarrow \ \beta_5\}) \\
= \ & \texttt{unify}\ (\{\beta_6 \rightarrow \beta_7 \ = \ \beta_8 \rightarrow \beta_8 , \\
& \qquad\qquad \beta_6 \rightarrow \beta_7 \ = \ \beta_5\})
\end{aligned}
$$

Algorithm J in action

$$
\begin{aligned}
& \texttt{unify}\ (\{(\beta_6 \to \beta_7) \to \beta_6 \to \beta_7 \ = \ (\beta_8 \to \beta_8) \ \to \ \beta_5\}) \\
= \ & \texttt{unify}\ (\{\beta_6 \to \beta_7 \ = \ \beta_8 \to \beta_8 , \\
& \qquad\quad\ \ \beta_6 \to \beta_7 \ = \ \beta_5\}) \\
= \ & \texttt{unify}\ (\{\beta_6 \ = \ \beta_8 , \\
& \qquad\quad\ \ \beta_7 \ = \ \beta_8 , \\
& \qquad\quad\ \ \beta_6 \to \beta_7 \ = \ \beta_5\})
\end{aligned}
$$

Algorithm J in action

$$
\begin{aligned}
& \texttt{unify}\ (\{(\beta_6 \to \beta_7) \to \beta_6 \to \beta_7 = (\beta_8 \to \beta_8) \to \beta_5\}) \\
= {} & \texttt{unify}\ (\{\beta_6 \to \beta_7 = \beta_8 \to \beta_8, \\
& \qquad\quad \beta_6 \to \beta_7 = \beta_5\}) \\
= {} & \texttt{unify}\ (\{\beta_6 = \beta_8, \\
& \qquad\quad \beta_7 = \beta_8, \\
& \qquad\quad \beta_6 \to \beta_7 = \beta_5\}) \\
= {} & \{\beta_6 \mapsto \beta_8,\ \beta_7 \mapsto \beta_8,\ \beta_5 \mapsto \beta_6 \to \beta_7\}
\end{aligned}
$$

Algorithm J in action

$$
\begin{array}{l}
J(\cdot,\ \textbf{let}\ \text{apply} = \lambda f.\lambda x.f\ x\ \textbf{in} \\
\quad \textbf{let}\ \text{id} = \lambda y.y\ \textbf{in} \\
\quad\quad \text{apply id}) = \\
\quad J(\cdot,\ \lambda f.\lambda x.f\ x) = (\beta_2 \to \beta_3) \to \beta_2 \to \beta_3 \\
\quad J(\cdot, \text{apply} : \forall \beta_2 \beta_3.(\beta_2 \to \beta_3) \to \beta_2 \to \beta_3, \\
\quad\quad \textbf{let}\ \text{id} = \lambda y.y\ \textbf{in}\ \text{apply id}) = \\
\quad\quad J(\cdot, \text{apply} : \forall \beta_2 \beta_3.(\beta_2 \to \beta_3) \to \beta_2 \to \beta_3, \\
\quad\quad\quad \lambda y.y) = \beta_4 \to \beta_4 \\
\quad\quad J(\cdot,\ \text{apply} : \forall \beta_2 \beta_3.(\beta_2 \to \beta_3) \to \beta_2 \to \beta_3, \text{id} : \forall \beta_4.\beta_4 \to \beta_4, \\
\quad\quad\quad \text{apply id}) = \beta_5 \\
\quad\quad\quad J(\cdot,\ \text{apply} : \forall \beta_2 \beta_3.(\beta_2 \to \beta_3) \to \beta_2 \to \beta_3, \\
\quad\quad\quad\quad\quad \text{id} : \forall \beta_4.\beta_4 \to \beta_4,\ \text{apply}) \\
\quad\quad\quad\quad = (\beta_6 \to \beta_7) \to \beta_6 \to \beta_7 \\
\quad\quad\quad J(\cdot,\ \text{apply} : \forall \beta_2 \beta_3.(\beta_2 \to \beta_3) \to \beta_2 \to \beta_3, \\
\quad\quad\quad\quad\quad \text{id} : \forall \beta_4.\beta_4 \to \beta_4,\ \text{id}) \\
\quad\quad\quad\quad = \beta_8 \to \beta_8
\end{array}
$$

Algorithm J in action

$$
\begin{aligned}
&J(\cdot,\ \mathbf{let}\ \mathsf{apply} = \lambda f.\lambda x.f\ x\ \mathbf{in}\\
&\quad \mathbf{let}\ \mathsf{id} = \lambda y.y\ \mathbf{in}\\
&\quad\quad \mathsf{apply}\ \mathsf{id}) =\\
&\quad J(\cdot,\ \lambda f.\lambda x.f\ x) = (\beta_2 \to \beta_3) \to \beta_2 \to \beta_3\\
&\quad J(\cdot, \mathsf{apply} : \forall \beta_2\beta_3.(\beta_2 \to \beta_3) \to \beta_2 \to \beta_3,\\
&\quad\quad \mathbf{let}\ \mathsf{id} = \lambda y.y\ \mathbf{in}\ \mathsf{apply}\ \mathsf{id}) =\\
&\quad\quad J(\cdot, \mathsf{apply} : \forall \beta_2\beta_3.(\beta_2 \to \beta_3) \to \beta_2 \to \beta_3,\\
&\quad\quad\quad \lambda y.y) = \beta_4 \to \beta_4\\
&\quad\quad J(\cdot,\ \mathsf{apply} : \forall \beta_2\beta_3.(\beta_2 \to \beta_3) \to \beta_2 \to \beta_3, \mathsf{id} : \forall \beta_4.\beta_4 \to \beta_4,\\
&\quad\quad\quad \mathsf{apply}\ \mathsf{id}) = \beta_8 \to \beta_8\\
&\quad\quad\quad J(\cdot,\ \mathsf{apply} : \forall \beta_2\beta_3.(\beta_2 \to \beta_3) \to \beta_2 \to \beta_3,\\
&\quad\quad\quad\quad\quad \mathsf{id} : \forall \beta_4.\beta_4 \to \beta_4,\ \mathsf{apply})\\
&\quad\quad\quad\quad = (\beta_8 \to \beta_8) \to \beta_8 \to \beta_8\\
&\quad\quad\quad J(\cdot,\ \mathsf{apply} : \forall \beta_2\beta_3.(\beta_2 \to \beta_3) \to \beta_2 \to \beta_3,\\
&\quad\quad\quad\quad\quad \mathsf{id} : \forall \beta_4.\beta_4 \to \beta_4,\ \mathsf{id})\\
&\quad\quad\quad\quad = \beta_8 \to \beta_8
\end{aligned}
$$

Algorithm J in action

$$
\begin{array}{l}
J(\cdot,\ \textbf{let}\ \mathsf{apply} = \lambda f.\lambda x.f\ x\ \textbf{in} \\
\quad \textbf{let}\ \mathsf{id} = \lambda y.y\ \textbf{in} \\
\quad\quad \mathsf{apply}\ \mathsf{id}) = \\
\quad J(\cdot,\ \lambda f.\lambda x.f\ x) = (\beta_2 \to \beta_3) \to \beta_2 \to \beta_3 \\
\quad J(\cdot, \mathsf{apply} : \forall \beta_2 \beta_3 . (\beta_2 \to \beta_3) \to \beta_2 \to \beta_3, \\
\quad\quad \textbf{let}\ \mathsf{id} = \lambda y.y\ \textbf{in}\ \mathsf{apply}\ \mathsf{id}) = \\
\quad\quad J(\cdot, \mathsf{apply} : \forall \beta_2 \beta_3 . (\beta_2 \to \beta_3) \to \beta_2 \to \beta_3, \\
\quad\quad\quad \lambda y.y) = \beta_4 \to \beta_4 \\
\quad\quad J(\cdot,\ \mathsf{apply} : \forall \beta_2 \beta_3 . (\beta_2 \to \beta_3) \to \beta_2 \to \beta_3, \mathsf{id} : \forall \beta_4 . \beta_4 \to \beta_4, \\
\quad\quad\quad \mathsf{apply}\ \mathsf{id}) = \beta_8 \to \beta_8
\end{array}
$$

Algorithm J in action

$$
\begin{array}{l}
J(\cdot,\ \textbf{let}\ apply = \lambda f.\lambda x.f\ x\ \textbf{in} \\
\quad\ \textbf{let}\ id = \lambda y.y\ \textbf{in} \\
\quad\quad apply\ id) = \\
\ J(\cdot,\ \lambda f.\lambda x.f\ x) = (\beta_2 \to \beta_3) \to \beta_2 \to \beta_3 \\
\ J(\cdot, apply : \forall \beta_2 \beta_3.(\beta_2 \to \beta_3) \to \beta_2 \to \beta_3, \\
\quad \textbf{let}\ id = \lambda y.y\ \textbf{in}\ apply\ id) = \beta_8 \to \beta_8 \\
\quad J(\cdot,\ apply : \forall \beta_2 \beta_3.(\beta_2 \to \beta_3) \to \beta_2 \to \beta_3, id : \forall \beta_4.\beta_4 \to \beta_4, \\
\quad\quad apply\ id) = \beta_8 \to \beta_8
\end{array}
$$

Algorithm J in action

```
J(·, let apply = λf.λx.f x in
       let id = λy.y in
        apply id) = β8 → β8
```

Type inference in practice

Type inference and recursion

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

Supporting imperative programming: the value restriction

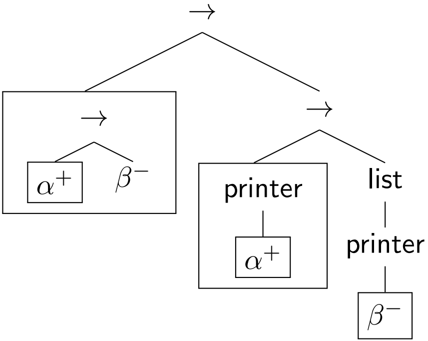
```
type 'a ref = { mutable contents : 'a }
val ref : 'a -> 'a ref
val ( ! ) : 'a ref -> 'a
val (:=) : 'a ref -> 'a -> unit

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

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

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