Last time

System Fω

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

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

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

This time

Assignment Project Exam Help

https://powcoder.com

F + M +

Add WeChat powcoder

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

$\forall \alpha . \alpha \rightarrow \alpha$

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

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

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

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

$\forall\alpha.\alpha \rightarrow \alpha$ ✓

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

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

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

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

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

$$\dfrac{\alpha \in \Gamma}{\Gamma \vdash \alpha \text{ is a type}}\ \alpha\text{-types} \qquad \dfrac{\Gamma \vdash A \text{ is a type} \quad \Gamma \vdash B \text{ is a type}}{\Gamma \vdash A \to B \text{ is a type}}\ \to\text{-types}$$

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

Environments

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

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

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

Typing rules for $\to$

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

Typing rules for schemes

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

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

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

$$a = b$$

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

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

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

Unification

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

$$\text{unify}(\emptyset) = []$$

$$\text{unify}(\{A = A\} \cup C) = \text{unify}(C)$$

$$\text{unify}(\{a = A\} \cup C) = \text{unify}([a \mapsto A]C) \circ [a \mapsto A]$$
$$\text{when } a \notin ftv(A)$$

$$\text{unify}(\{A = a\} \cup C) = \text{unify}([a \mapsto A]C) \circ [a \mapsto A]$$
$$\text{when } a \notin ftv(A)$$

$$\text{unify}(\{A \to B = A' \to B'\} \cup C) = \text{unify}(\{A = A', B = B'\} \cup C)$$

$$\text{unify}(\{A = B\} \cup C) = FAIL$$

Algorithm J

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

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

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

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

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

Algorithm J in action

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

Algorithm J in action

```
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:b_1, 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) = b1 → b2 → b3
    J(·,f:b1, λx.f x) = b2 → b3
      J(·,f:b1,x:b2, f x) = b3
```

Algorithm J in action

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

Algorithm J in action

J(·, let apply = λf.λx.f x in
let id = λy.y in
apply id) =
J(·, λf.λx.f x) = $b_1 \to b_2 \to b_3$
J(·,f:b_1, λx.f x) = $b_2 \to b_3$
J(·,f:b_1,x:b_2, f x) = b_3
J(·,f:b_1,x:b_2, f) = b_1

Algorithm J in action

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

Algorithm J in action

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

Algorithm J in action

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

Algorithm J in action

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

Algorithm J in action

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

Algorithm J in action

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

Algorithm J in action

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

Algorithm J in action

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

Algorithm J in action

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

Algorithm J in action

J($\cdot$, let apply = λf.λx.f x in
    let id = λy.y in
    apply id) =

J($\cdot$, λf.λx.f x) = $(b_2 \to b_3) \to b_2 \to b_3$

J($\cdot$,apply:$\forall\alpha_2\alpha_3.(\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3$,
  let id = λy.y in apply id) =

J($\cdot$,apply:$\forall\alpha_2\alpha_3.(\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3$,
  λy.y) = $b_4 \to b_4$

ftv($b_4 \to b_4$) = $\{b_4\}$

ftv($\cdot$,apply:$\forall\alpha_2\alpha_3.(\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3$) = {}

$\{b_4\}$ \ {} = $\{b_4\}$

Algorithm J in action

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

Algorithm J in action

$\texttt{J}(\cdot,\ \texttt{let apply = }\lambda\texttt{f.}\lambda\texttt{x.f x in}$

$\quad\texttt{let id = }\lambda\texttt{y.y in}$

$\quad\quad\texttt{apply id) =}$

$\texttt{J}(\cdot,\ \lambda\texttt{f.}\lambda\texttt{x.f x}) = (b_2 \to b_3) \to b_2 \to b_3$

$\texttt{J}(\cdot,\texttt{apply}:\forall\alpha_2\alpha_3.(\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3,$

$\quad\texttt{let id = }\lambda\texttt{y.y in apply id) =}$

$\quad\texttt{J}(\cdot,\texttt{apply}:\forall\alpha_2\alpha_3.(\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3,$

$\quad\quad\lambda\texttt{y.y}) = b_4 \to b_4$

$\quad\texttt{J}(\cdot,\ \texttt{apply}:\forall\alpha_2\alpha_3.(\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3,\texttt{id}:\forall\alpha_4.\alpha_4 \to \alpha_4,$

$\quad\quad\texttt{apply id}) = b_5$

$\quad\quad\texttt{J}(\cdot,\ \texttt{apply}:\forall\alpha_2\alpha_3.(\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3,$

$\quad\quad\quad\quad\texttt{id}:\forall\alpha_4.\alpha_4 \to \alpha_4,\ \texttt{apply})$

$\quad\quad\quad = (b_6 \to b_7) \to b_6 \to b_7$

Algorithm J in action

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

J(·, λf.λx.f x) = $(b_2 \to b_3) \to b_2 \to b_3$

J(·,apply:$\forall \alpha_2 \alpha_3.(\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3$,
  let id = λy.y in apply id) =
  J(·,apply:$\forall \alpha_2 \alpha_3.(\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3$,
   λy.y) = $b_4 \to b_4$
  J(·, apply:$\forall \alpha_2 \alpha_3.(\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3$, id:$\forall \alpha_4.\alpha_4 \to \alpha_4$,
   apply id) = b_5
   J(·, apply:$\forall \alpha_2 \alpha_3.(\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3$,
     id:$\forall \alpha_4.\alpha_4 \to \alpha_4$, apply)
    = $(b_6 \to b_7) \to b_6 \to b_7$
   J(·, apply:$\forall \alpha_2 \alpha_3.(\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3$,
     id:$\forall \alpha_4.\alpha_4 \to \alpha_4$, id)
    = $b_8 \to b_8$

Algorithm J in action

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

Algorithm J in action


```
  unify ({(b6 → b7) → b6 → b7  =  (b8 → b8)  →  b5})
= unify ({b6 → b7  =  b8 → b8,
          b6 → b7  =  b5})
```

Algorithm J in action

$$
\begin{array}{l}
\texttt{unify}\ (\{(b_6 \rightarrow b_7) \rightarrow b_6 \rightarrow b_7 = (b_8 \rightarrow b_8) \rightarrow b_5\}) \\
= \texttt{unify}\ (\{b_6 \rightarrow b_7 = b_8 \rightarrow b_8, \\
\qquad\qquad\quad b_6 \rightarrow b_7 = b_5\}) \\
= \texttt{unify}\ (\{b_6 = b_8, \\
\qquad\qquad\quad b_7 = b_8, \\
\qquad\qquad\quad b_6 \rightarrow b_7 = b_5\})
\end{array}
$$

Algorithm J in action

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

Algorithm J in action

J(·, let apply = λf.λx.f x in
let id = λy.y in
apply id) =
J(·, λf.λx.f x) = $(b_2 \to b_3) \to b_2 \to b_3$
J(·, apply : $\forall \alpha_2 \alpha_3 . (\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3$,
let id = λy.y in apply id) =
J(·, apply : $\forall \alpha_2 \alpha_3 . (\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3$,
λy.y) = $b_4 \to b_4$
J(·, apply : $\forall \alpha_2 \alpha_3 . (\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3$, id : $\forall \alpha_4 . \alpha_4 \to \alpha_4$,
apply id) = b_5
J(·, apply : $\forall \alpha_2 \alpha_3 . (\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3$,
id : $\forall \alpha_4 . \alpha_4 \to \alpha_4$, apply)
= $(b_6 \to b_7) \to b_6 \to b_7$
J(·, apply : $\forall \alpha_2 \alpha_3 . (\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3$,
id : $\forall \alpha_4 . \alpha_4 \to \alpha_4$, id)
= $b_8 \to b_8$

Algorithm J in action

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

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) = b₈ → b₈
```

Algorithm J in action

$\texttt{J}(\cdot,\ \texttt{let apply} = \lambda \texttt{f}.\lambda \texttt{x}.\texttt{f x in}$

$\quad \texttt{let id} = \lambda \texttt{y}.\texttt{y in}$

$\quad \texttt{apply id}) =$

$\quad \texttt{J}(\cdot,\ \lambda \texttt{f}.\lambda \texttt{x}.\texttt{f x}) = (b_2 \to b_3) \to b_2 \to b_3$

$\quad \texttt{J}(\cdot, \texttt{apply}: \forall \alpha_2 \alpha_3 . (\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3,$

$\quad\quad \texttt{let id} = \lambda \texttt{y}.\texttt{y in apply id}) = b_8 \to b_8$

$\quad\quad \texttt{J}(\cdot,\ \texttt{apply}: \forall \alpha_2 \alpha_3 . (\alpha_2 \to \alpha_3) \to \alpha_2 \to \alpha_3, \texttt{id}: \forall \alpha_4 . \alpha_4 \to \alpha_4,$

$\quad\quad\quad \texttt{apply id}) = b_8 \to b_8$

Algorithm J in action

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

Type inference in practice

Type inference and recursion

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

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
```

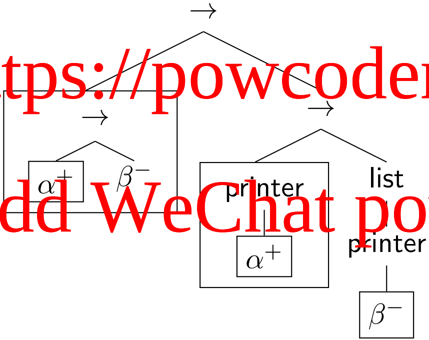


```
let r = ref None in
  r := Some "boom";
  match !r with
    None -> ()
  | Some f -> f ()
```

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

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