# Specialization

**Concepts of Programming Languages**
**Lecture 24**

CAS CS 320

## Outline

- » Discuss **specialization** and how it relates to principle types
- » Demo an implementation of **constraint-based type inference**
- » Put the finishing touches on our discussion of type inference

## Recap

## Recall: Principle Types

$$\Gamma \vdash e : \tau \dashv \mathscr{C}$$

## Recall: Principle Types

$$\Gamma \vdash e : \tau \dashv \mathcal{C}$$

The constraints $\mathcal{C}$ defined a *unification problem*. Given a most general unifier $\mathcal{S}$ we can get the "actual" type of $e$:

## Recall: Principle Types

$$\Gamma \vdash e : \tau \dashv \mathscr{C}$$

The constraints $\mathscr{C}$ defined a *unification problem*. Given a most general unifier $\mathscr{S}$ we can get the "actual" type of $e$:

$$\mathsf{principle}(\tau, \mathscr{C}) = \forall \alpha_1 \ldots \forall \alpha_k . \mathscr{S}\tau \quad \texttt{where} \quad \mathsf{FV}(\mathscr{S}\tau) = \{\alpha_1, \ldots, \alpha_k\}$$

## Recall: Principle Types

$$\Gamma \vdash e : \tau \dashv \mathscr{C}$$

The constraints $\mathscr{C}$ defined a *unification problem*. Given a most general unifier $\mathscr{S}$ we can get the "actual" type of $e$:

$$\text{principle}(\tau, \mathscr{C}) = \forall \alpha_1 \ldots \forall \alpha_k . \mathscr{S}\tau \quad \text{where} \quad \text{FV}(\mathscr{S}\tau) = \{\alpha_1, \ldots, \alpha_k\}$$

i.e, the **principle type** of $e$ (<u>note:</u> it may not exist). Every type we *could* give $e$ is a *specialization* of $\forall \alpha_1, \ldots, \alpha_k . \mathscr{S}\tau$

## Recall: Putting everything together

## Recall: Putting everything together

`input:` `program` $P$ `(sequence of top-level let-expressions)`

# Recall: Putting everything together

```
input: program $P$ (sequence of top-level let-expressions)

$\Gamma \leftarrow \varnothing$
```

# Recall: Putting everything together

```
input: program $P$ (sequence of top-level let-expressions)

$\Gamma \leftarrow \varnothing$

FOR EACH top-level let-expression $\text{let } x = e$ in $P$:
```

# Recall: Putting everything together

```
input: program $P$ (sequence of top-level let-expressions)

$\Gamma \leftarrow \varnothing$

FOR EACH top-level let-expression $\text{let } x = e$ in $P$:

    1. Constraint-based inference: Determine $\tau$ and $\mathscr{C}$ such that $\Gamma \vdash e : \tau \dashv \mathscr{C}$ is
       derivable
```

# Recall: Putting everything together

input: program $P$ (sequence of top-level let-expressions)

$\Gamma \leftarrow \varnothing$

**FOR EACH** top-level let-expression $\text{let } x = e$ in $P$:

1. *Constraint-based inference:* Determine $\tau$ and $\mathscr{C}$ such that $\Gamma \vdash e : \tau \dashv \mathscr{C}$ is derivable

2. *Unification:* Solve $\mathscr{C}$ to get a most general unifier $\mathscr{S}$ (**TYPE ERROR** if this fails)

# Recall: Putting everything together

input: program $P$ (sequence of top-level let-expressions)

$\Gamma \leftarrow \varnothing$

**FOR EACH** top-level let-expression $\mathsf{let}\ x = e$ in $P$:

1. *Constraint-based inference:* Determine $\tau$ and $\mathscr{C}$ such that $\Gamma \vdash e : \tau \dashv \mathscr{C}$ is derivable
2. *Unification:* Solve $\mathscr{C}$ to get a most general unifier $\mathscr{S}$ (**TYPE ERROR** if this fails)
3. *Generalization:* Quantify over the free variables in $\mathscr{S}\tau$ to get the principle type $\forall \alpha_1 \ldots \forall \alpha_k . \mathscr{S}\tau$ of $e$

# Recall: Putting everything together

```
input: program $P$ (sequence of top-level let-expressions)

$\Gamma \leftarrow \varnothing$

FOR EACH top-level let-expression $\text{let } x = e$ in $P$:

    1. Constraint-based inference: Determine $\tau$ and $\mathscr{C}$ such that $\Gamma \vdash e : \tau \dashv \mathscr{C}$ is
       derivable

    2. Unification: Solve $\mathscr{C}$ to get a most general unifier $\mathscr{S}$ (TYPE ERROR if
       this fails)

    3. Generalization: Quantify over the free variables in $\mathscr{S}\tau$ to get the
       principle type $\forall \alpha_1 \ldots \forall \alpha_k . \mathscr{S}\tau$ of $e$

    4. Add $(x : \forall \alpha_1 \ldots \forall \alpha_k . \mathscr{S}\tau)$ to $\Gamma$
```

## Example

*Determine the principle type of* $\lambda f.\lambda x.f\,x+1$

## Example

*Show that* $\mathsf{let}\ f = \lambda x\,.\,x\ \mathsf{in}\ f\ (f\ 2 = 2)$ *has no principle type*

## Specialization

## Recall: HM$^-$ (Syntax)

$$
\begin{aligned}
e ::= {} & \lambda x\,.\,e \mid ee \\
& \mid \text{let } x = e \text{ in } e \\
& \mid \text{if } e \text{ then } e \text{ else } e \\
& \mid e + e \mid e = e \\
& \mid n \mid x \\
\sigma ::= {} & \text{int} \mid \text{bool} \mid \alpha \mid \sigma \to \sigma \\
\tau ::= {} & \sigma \mid \forall \alpha\,.\,\tau
\end{aligned}
$$

# Recall: HM$^-$ (Typing)

$$\frac{n \texttt{ is an integer}}{\Gamma \vdash n : \mathsf{int} \dashv \varnothing}\ (\texttt{int}) \qquad \frac{\Gamma \vdash e_1 : \tau_1 \dashv \mathscr{C}_1 \qquad \Gamma \vdash e_2 : \tau_2 \dashv \mathscr{C}_2 \qquad \Gamma \vdash e_3 : \tau_3 \dashv \mathscr{C}_3}{\Gamma \vdash \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 : \tau_3 \dashv \tau_1 \doteq \mathsf{bool}, \tau_2 \doteq \tau_3, \mathscr{C}_1, \mathscr{C}_2, \mathscr{C}_3}\ (\texttt{if})$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \dashv \mathscr{C}_1 \qquad \Gamma \vdash e_2 : \tau_2 \dashv \mathscr{C}_2}{\Gamma \vdash e_1 = e_2 : \mathsf{bool} \dashv \tau_1 \doteq \tau_2, \mathscr{C}_1, \mathscr{C}_2}\ (\texttt{eq}) \qquad \frac{\Gamma \vdash e_1 : \tau_1 \dashv \mathscr{C}_1 \qquad \Gamma \vdash e_2 : \tau_2 \dashv \mathscr{C}_2}{\Gamma \vdash e_1 + e_2 : \mathsf{int} \dashv \tau_1 \doteq \mathsf{int}, \tau_2 \doteq \mathsf{int}, \mathscr{C}_1, \mathscr{C}_2}\ (\texttt{add})$$

$$\frac{\alpha \texttt{ is fresh} \qquad \Gamma, x : \alpha \vdash e : \tau \dashv \mathscr{C}}{\Gamma \vdash \lambda x . e : \alpha \to \tau \dashv \mathscr{C}}\ (\texttt{fun})$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \dashv \mathscr{C}_1 \qquad \Gamma \vdash e_2 : \tau_2 \dashv \mathscr{C}_2 \qquad \alpha \texttt{ is fresh}}{\Gamma \vdash e_1 e_2 : \alpha \dashv \tau_1 \doteq \tau_2 \to \alpha, \mathscr{C}_1, \mathscr{C}_2}\ (\texttt{app})$$

## Recall: HM$^-$ (Typing Variables)

$$\frac{(x : \forall \alpha_1 . \forall \alpha_2 \ldots \forall \alpha_k . \tau) \in \Gamma \qquad \beta_1, \ldots, \beta_k \text{ are fresh}}{\Gamma \vdash x : [\beta_1/\alpha_1]\ldots[\beta_k/\alpha_k]\tau \dashv \varnothing} \text{(var)}$$

If $x$ is declared in $\Gamma$, then $x$ can be given the type $\tau$ *with all free variables replaced by **fresh variables***

*This is where the polymorphism magic happens*

**Fresh variables can be unified with anything**

# An Alternative Formulation

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

It's possible to give a type system for $\text{HM}^-$ *without* constraints

It's very similar to our 320Caml system, but with some rules for dealing with **quantification** and **specialization**

# HM$^-$ (Alternative Typing)

$$\frac{n \texttt{ is an integer}}{\Gamma \vdash n : \mathsf{int}}\ \texttt{(int)}$$

$$\frac{\Gamma \vdash e_1 : \mathsf{bool} \qquad \Gamma \vdash e_2 : \tau \qquad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 : \tau}\ \texttt{(if)}$$

$$\frac{\Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1 = e_2 : \mathsf{bool}}\ \texttt{(eq)}$$

$$\frac{\Gamma \vdash e_1 : \mathsf{int} \qquad \Gamma \vdash e_2 : \mathsf{int}}{\Gamma \vdash e_1 + e_2 : \mathsf{int}}\ \texttt{(add)}$$

$$\frac{\Gamma, x : \tau_1 \vdash e : \tau_2 \dashv \mathscr{C}}{\Gamma \vdash \lambda x \,.\, e : \tau_1 \to \tau_2 \dashv \mathscr{C}}\ \texttt{(fun)}$$

$$\frac{\Gamma \vdash e_1 : \tau_2 \to \tau \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1 e_2 : \tau}\ \texttt{(app)}$$

$$\frac{\tau_1 \texttt{ is a monotype} \qquad \Gamma \vdash e_1 : \tau_1 \qquad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : \tau_2 \dashv \mathscr{C}_1, \mathscr{C}_2}\ \texttt{(let)}$$

# HM$^-$ (Alternative Typing)

familiar rules

$$\frac{n \texttt{ is an integer}}{\Gamma \vdash n : \mathsf{int}}\ (\texttt{int}) \qquad \frac{\Gamma \vdash e_1 : \mathsf{bool} \quad \Gamma \vdash e_2 : \tau \quad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 : \tau}\ (\texttt{if})$$

$$\frac{\Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1 = e_2 : \mathsf{bool}}\ (\texttt{eq}) \qquad \frac{\Gamma \vdash e_1 : \mathsf{int} \quad \Gamma \vdash e_2 : \mathsf{int}}{\Gamma \vdash e_1 + e_2 : \mathsf{int}}\ (\texttt{add})$$

$$\frac{\Gamma, x : \tau_1 \vdash e : \tau_2 \dashv \mathscr{C}}{\Gamma \vdash \lambda x . e : \tau_1 \to \tau_2 \dashv \mathscr{C}}\ (\texttt{fun}) \qquad \frac{\Gamma \vdash e_1 : \tau_2 \to \tau \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1 e_2 : \tau}\ (\texttt{app})$$

$$\frac{\tau_1 \texttt{ is a monotype} \quad \Gamma \vdash e_1 : \tau_1 \quad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : \tau_2 \dashv \mathscr{C}_1, \mathscr{C}_2}\ (\texttt{let})$$

# Generalization and Specialization

$$\frac{\Gamma \vdash e : \tau \qquad \alpha \texttt{ not free in } \Gamma}{\Gamma \vdash e : \forall \alpha . \tau}\ \texttt{(gen)} \qquad \frac{(x : \tau) \in \Gamma \qquad \tau \sqsubseteq \tau'}{\Gamma \vdash x : \tau'}\ \texttt{(var)}$$

## Generalization and Specialization

$$\frac{\Gamma \vdash e : \tau \qquad \alpha \text{ not free in } \Gamma}{\Gamma \vdash e : \forall \alpha . \tau} \text{ (gen)} \qquad \frac{(x : \tau) \in \Gamma \qquad \tau \sqsubseteq \tau'}{\Gamma \vdash x : \tau'} \text{ (var)}$$

The generalization rule is like the one from System F

## Generalization and Specialization

$$\frac{\Gamma \vdash e : \tau \qquad \alpha \text{ not free in } \Gamma}{\Gamma \vdash e : \forall \alpha . \tau} \text{ (gen)} \qquad \frac{(x : \tau) \in \Gamma \qquad \tau \sqsubseteq \tau'}{\Gamma \vdash x : \tau'} \text{ (var)}$$

The generalization rule is like the one from System F

The main difference: we introduce a notion of **specialization** which allows us to *instantiate* polymorphic functions at particular types

# Generalization and Specialization

$$\frac{\Gamma \vdash e : \tau \qquad \alpha \texttt{ not free in } \Gamma}{\Gamma \vdash e : \forall \alpha . \tau} \text{ (gen)} \qquad \frac{(x : \tau) \in \Gamma \qquad \tau \sqsubseteq \tau'}{\Gamma \vdash x : \tau'} \text{ (var)}$$

The generalization rule is like the one from System F

The main difference: we introduce a notion of **specialization** which allows us to *instantiate* polymorphic functions at particular types

"$\sqsubseteq$" defined a *partial order* on type schemes

## Specialization (Informal)

$$\forall \alpha_1 \ldots \forall \alpha_m . \tau \sqsubseteq \forall \beta_1 \ldots \forall \beta_n . \tau'$$

A type scheme $T_2$ **specializes** $T_1$, written $T_1 \sqsubseteq T_2$ if $T_2$ the result of instantiating the bound variables of $T_1$ and generalizing over some of the variables introduced by the instantiation

## Specialization (Formal)

$$\frac{\tau_1, \ldots, \tau_m \text{ are monotypes} \quad \tau' = [\tau_m/\alpha_m]\ldots[\tau_1/\alpha_1]\tau \quad \beta_1, \ldots, \beta_n \notin \mathsf{FV}(\tau) \setminus \{\alpha_1, \ldots, \alpha_m\}}{\forall \alpha_1 \ldots \forall \alpha_m \,.\, \tau \sqsubseteq \forall \beta_1 \ldots \forall \beta_n \,.\, \tau'}$$

A *specialization* of a type scheme is an instantiation of its bound variable, together with some generalizations over remaining free variables

## Examples

## Examples

$$\begin{aligned}\forall\alpha\,.\,\forall\beta\,.\,\alpha \to \beta \to \alpha &\sqsubseteq \forall\eta\,.\,\eta \to \mathsf{bool} \to \eta \\ &\sqsubseteq \mathsf{int} \to \mathsf{bool} \to \mathsf{int}\end{aligned}$$

## Examples

$$\forall\alpha\,.\,\forall\beta\,.\,\alpha \to \beta \to \alpha \sqsubseteq \forall\eta\,.\,\eta \to \mathsf{bool} \to \eta$$
$$\sqsubseteq \mathsf{int} \to \mathsf{bool} \to \mathsf{int}$$

$$\forall\alpha\,.\,\forall\beta\,.\,\alpha \to \beta \to \alpha \sqsubseteq \forall\gamma\,.\,\mathsf{bool} \to (\gamma \to \gamma) \to \mathsf{bool}$$
$$\sqsubseteq \mathsf{bool} \to (\mathsf{int} \to \mathsf{int}) \to \mathsf{bool}$$

## Examples

$$\begin{aligned}\forall\alpha\,.\,\forall\beta\,.\,\alpha \to \beta \to \alpha &\sqsubseteq \forall\eta\,.\,\eta \to \mathsf{bool} \to \eta \\ &\sqsubseteq \mathsf{int} \to \mathsf{bool} \to \mathsf{int}\end{aligned}$$

$$\begin{aligned}\forall\alpha\,.\,\forall\beta\,.\,\alpha \to \beta \to \alpha &\sqsubseteq \forall\gamma\,.\,\mathsf{bool} \to (\gamma \to \gamma) \to \mathsf{bool} \\ &\sqsubseteq \mathsf{bool} \to (\mathsf{int} \to \mathsf{int}) \to \mathsf{bool}\end{aligned}$$

$$\begin{aligned}\forall\alpha\,.\,\forall\beta\,.\,\alpha \to \beta \to \alpha &\sqsubseteq \mathsf{bool} \to (\gamma \to \gamma) \to \mathsf{bool} \\ &\not\sqsubseteq \mathsf{bool} \to (\mathsf{int} \to \mathsf{int}) \to \mathsf{bool}\end{aligned}$$

## Specialization and Principle Types

## Specialization and Principle Types

Theorem. If $\Gamma \vdash e : \tau'$ then there is a type $\tau$ and constraints $\mathscr{C}$ such that $\Gamma \vdash e : \tau \dashv \mathscr{C}$ and principle$(\tau, \mathscr{C}) \sqsubseteq \tau'$

## Specialization and Principle Types

<u>Theorem.</u> If $\Gamma \vdash e : \tau'$ then there is a type $\tau$ and constraints $\mathscr{C}$ such that $\Gamma \vdash e : \tau \dashv \mathscr{C}$ and $\mathsf{principle}(\tau, \mathscr{C}) \sqsubseteq \tau'$

<u>Theorem.</u> If $\Gamma \vdash e : \tau \dashv \mathscr{C}$ and $\mathsf{principle}(\tau, \mathscr{C}) \sqsubseteq \tau'$ then $\Gamma \vdash e : \tau'$

## Specialization and Principle Types

Theorem. If $\Gamma \vdash e : \tau'$ then there is a type $\tau$ and constraints $\mathscr{C}$ such that $\Gamma \vdash e : \tau \dashv \mathscr{C}$ and $\mathsf{principle}(\tau, \mathscr{C}) \sqsubseteq \tau'$

Theorem. If $\Gamma \vdash e : \tau \dashv \mathscr{C}$ and $\mathsf{principle}(\tau, \mathscr{C}) \sqsubseteq \tau'$ then $\Gamma \vdash e : \tau'$

*The principle type is the most general "lowest" type with respect to specialization*

## Example

$\{f : \forall\alpha\,.\,\alpha \to \alpha\} \vdash f\ (f\ 2 = 2) : \mathsf{bool}$

## Why use constraints at all?

$$\frac{(x : \tau) \in \Gamma \qquad \tau \sqsubseteq \tau'}{\Gamma \vdash x : \tau'}\ \texttt{(var)} \qquad \frac{(x : \forall \alpha_1 . \forall \alpha_2 \ldots \forall \alpha_k . \tau) \in \Gamma \qquad \beta_1, \ldots, \beta_k \ \texttt{are fresh}}{\Gamma \vdash x : [\beta_1/\alpha_1] \ldots [\beta_k/\alpha_k]\tau \dashv \varnothing}\ \texttt{(var)}$$

## Why use constraints at all?

$$\frac{(x : \tau) \in \Gamma \qquad \tau \sqsubseteq \tau'}{\Gamma \vdash x : \tau'}\ \texttt{(var)} \qquad \frac{(x : \forall \alpha_1 . \forall \alpha_2 \ldots \forall \alpha_k . \tau) \in \Gamma \qquad \beta_1, \ldots, \beta_k \text{ are fresh}}{\Gamma \vdash x : [\beta_1/\alpha_1] \ldots [\beta_k/\alpha_k]\tau \dashv \varnothing}\ \texttt{(var)}$$

The alternative type rules are theoretically nice but not *algorithmic*

# Why use constraints at all?

$$\frac{(x : \tau) \in \Gamma \qquad \tau \sqsubseteq \tau'}{\Gamma \vdash x : \tau'} \text{ (var)} \qquad \frac{(x : \forall \alpha_1 . \forall \alpha_2 \ldots \forall \alpha_k . \tau) \in \Gamma \qquad \beta_1, \ldots, \beta_k \text{ are fresh}}{\Gamma \vdash x : [\beta_1/\alpha_1]\ldots[\beta_k/\alpha_k]\tau \dashv \varnothing} \text{ (var)}$$

The alternative type rules are theoretically nice but not *algorithmic*

*How do I choose which specialization to use in a derivation?*

# Why use constraints at all?

$$\frac{(x : \tau) \in \Gamma \qquad \tau \sqsubseteq \tau'}{\Gamma \vdash x : \tau'} \ (\texttt{var}) \qquad \frac{(x : \forall \alpha_1 . \forall \alpha_2 \ldots \forall \alpha_k . \tau) \in \Gamma \qquad \beta_1, \ldots, \beta_k \ \texttt{are fresh}}{\Gamma \vdash x : [\beta_1/\alpha_1] \ldots [\beta_k/\alpha_k]\tau \dashv \varnothing} \ (\texttt{var})$$

The alternative type rules are theoretically nice but not *algorithmic*

*How do I choose which specialization to use in a derivation?*

Constraints allow us to determine *which* specializations we should use *after the fact*

## demo

(constraint-based inference)

## HM$^{-}$ (Typing Integers)

$$\frac{n \texttt{ is an integer}}{\Gamma \vdash n : \mathsf{int} \dashv \varnothing}\ \texttt{(int)}$$

## Recall: HM$^-$ (Typing Addition)

$$\frac{\Gamma \vdash e_1 : \tau_1 \dashv \mathscr{C}_1 \qquad \Gamma \vdash e_2 : \tau_2 \dashv \mathscr{C}_2}{\Gamma \vdash e_1 + e_2 : \mathsf{int} \dashv \tau_1 \doteq \mathsf{int}, \tau_2 \doteq \mathsf{int}, \mathscr{C}_1, \mathscr{C}_2} \ (\mathsf{add})$$

## Recall: HM$^-$ (Typing Equality)

$$\frac{\Gamma \vdash e_1 : \tau_1 \dashv \mathscr{C}_1 \qquad \Gamma \vdash e_2 : \tau_2 \dashv \mathscr{C}_2}{\Gamma \vdash e_1 = e_2 : \textsf{bool} \dashv \tau_1 \doteq \tau_2, \mathscr{C}_1, \mathscr{C}_2} \ (\text{eq})$$

## Recall: HM⁻ (Typing If-Expressions)

$$\frac{\Gamma \vdash e_1 : \tau_1 \dashv \mathscr{C}_1 \qquad \Gamma \vdash e_2 : \tau_2 \dashv \mathscr{C}_2 \qquad \Gamma \vdash e_3 : \tau_3 \dashv \mathscr{C}_3}{\Gamma \vdash \textsf{if } e_1 \textsf{ then } e_2 \textsf{ else } e_3 : \tau_3 \dashv \tau_1 \doteq \textsf{bool}, \tau_2 \doteq \tau_3, \mathscr{C}_1, \mathscr{C}_2, \mathscr{C}_3} \; (\texttt{if})$$

## HM$^-$ (Typing Let-Expressions)

$$\frac{\Gamma \vdash e_1 : \tau_1 \dashv \mathscr{C}_1 \qquad \Gamma, x : \tau_1 \vdash e_2 : \tau_2 \dashv \mathscr{C}_2}{\Gamma \vdash \textsf{let } x = e_1 \textsf{ in } e_2 : \tau_2 \dashv \mathscr{C}_1, \mathscr{C}_2} \ \texttt{(let)}$$

## Recall: HM⁻ (Typing Functions)

$$\frac{\alpha \texttt{ is fresh} \qquad \Gamma, x : \alpha \vdash e : \tau \dashv \mathscr{C}}{\Gamma \vdash \lambda x \,.\, e : \alpha \to \tau \dashv \mathscr{C}} \; \texttt{(fun)}$$

## Recall: HM⁻ (Typing Applications)

$$\frac{\Gamma \vdash e_1 : \tau_1 \dashv \mathscr{C}_1 \qquad \Gamma \vdash e_2 : \tau_2 \dashv \mathscr{C}_2 \qquad \alpha \texttt{ is fresh}}{\Gamma \vdash e_1 e_2 : \alpha \dashv \tau_1 \doteq \tau_2 \to \alpha, \mathscr{C}_1, \mathscr{C}_2} \; \texttt{(app)}$$

## Recall: HM$^-$ (Typing Variables)

$$\frac{(x : \forall \alpha_1 . \forall \alpha_2 \ldots \forall \alpha_k . \tau) \in \Gamma \qquad \beta_1, \ldots, \beta_k \texttt{ are fresh}}{\Gamma \vdash x : [\beta_1/\alpha_1]\ldots[\beta_k/\alpha_k]\tau \dashv \varnothing} \ \texttt{(var)}$$

## Summary

The **principle type** of an expression is the most general type we could give it

**Specialization** defines a partial ordering on type schemes from most to least general

Our unification algorithm gives us a most general unifier, which will always give us the principle type of an expression