## Programming Languages and Compilers (CS 421)

Elsa L Gunter
2112 SC, UIUC
http://courses.engr.illinois.edu/cs421

Based in part on slides by Mattox Beckman, as updated by Vikram Adve and Gul Agha



## Review: In Class Activity

ACT 4



## Mea Culpa

- The above system can't handle polymorphism as in OCAML
- No type variables in type language (only meta-variable in the logic)
- Would need:
  - Object level type variables and some kind of type quantification
  - **let** and **let rec** rules to introduce polymorphism
  - Explicit rule to eliminate (instantiate) polymorphism



## Support for Polymorphic Types

- Monomorpic Types ($\tau$):
  - Basic Types: int, bool, float, string, unit, …
  - Type Variables: $\alpha$, $\beta$, $\gamma$, $\delta$, $\varepsilon$
  - Compound Types: $\alpha \rightarrow \beta$, int * string, bool list, …
- Polymorphic Types:
  - Monomorphic types $\tau$
  - Universally quantified monomorphic types
  - $\forall \alpha_1, \ldots, \alpha_n . \tau$
  - Can think of $\tau$ as same as $\forall . \tau$



## Example FreeVars Calculations

- Vars('a -> (int -> 'b) -> 'a) ={'a , 'b}
- FreeVars (All 'b. 'a -> (int -> 'b) -> 'a) =
- {'a , 'b} – {'b}= {'a}
- FreeVars {
- x : All 'b. 'a -> (int -> 'b) -> 'a,
- id: All 'c. 'c -> 'c,
- y: All 'c. 'a -> ('b -> 'c)
- } = {'a} U {} U {'a, 'b} = {'a, 'b}



## Support for Polymorphic Types

- Typing Environment $\Gamma$ supplies polymorphic types (which will often just be monomorphic) for variables
- Free variables of monomorphic type just type variables that occur in it
  - Write FreeVars($\tau$)
- Free variables of polymorphic type removes variables that are universally quantified
  - FreeVars($\forall \alpha_1, \ldots, \alpha_n . \tau$) = FreeVars($\tau$) – $\{\alpha_1, \ldots, \alpha_n\}$
- FreeVars($\Gamma$) = all FreeVars of types in range of $\Gamma$

## Monomorphic to Polymorphic

- Given:
  - type environment $\Gamma$
  - monomorphic type $\tau$
  - $\tau$ shares type variables with $\Gamma$
- Want most polymorphic type for $\tau$ that doesn't break sharing type variables with $\Gamma$
- $Gen(\tau, \Gamma) = \forall \alpha_1, \ldots, \alpha_n . \tau$ where $\{\alpha_1, \ldots, \alpha_n\} = freeVars(\tau) - freeVars(\Gamma)$



## Polymorphic Typing Rules

- A *type judgement* has the form
  $\Gamma$ |- exp : $\tau$
  - $\Gamma$ uses polymorphic types
  - $\tau$ still monomorphic
- Most rules stay same (except use more general typing environments)
- Rules that change:
  - Variables
  - Let and Let Rec
  - Allow polymorphic constants
- Worth noting functions again



## Polymorphic Let and Let Rec

- let rule:

$$\frac{\Gamma \mid\!- e_1 : \tau_1 \quad \{x : Gen(\tau_1,\Gamma)\} + \Gamma \mid\!- e_2 : \tau_2}{\Gamma \mid\!- (\text{let } x = e_1 \text{ in } e_2) : \tau_2}$$

- let rec rule:

$$\frac{\{x : \tau_1\} + \Gamma \mid\!- e_1 : \tau_1 \quad \{x : Gen(\tau_1,\Gamma)\} + \Gamma \mid\!- e_2 : \tau_2}{\Gamma \mid\!- (\text{let rec } x = e_1 \text{ in } e_2) : \tau_2}$$



## Polymorphic Variables (Identifiers)

Variable axiom:

$$\frac{}{\Gamma \mid\!- x : \varphi(\tau)} \quad \text{if } \Gamma(x) = \forall \alpha_1, \ldots, \alpha_n . \tau$$

- Where $\varphi$ replaces all occurrences of $\alpha_1, \ldots, \alpha_n$ by monotypes $\tau_1, \ldots, \tau_n$
- Note: Monomorphic rule special case:

$$\frac{}{\Gamma \mid\!- x : \tau} \quad \text{if } \Gamma(x) = \tau$$

- Constants treated same way



## Fun Rule Stays the Same

- fun rule:

$$\frac{\{x : \tau_1\} + \Gamma \mid\!- e : \tau_2}{\Gamma \mid\!- \text{fun } x \text{ -> } e : \tau_1 \to \tau_2}$$

- Types $\tau_1, \tau_2$ monomorphic
- Function argument must always be used at same type in function body



## Polymorphic Example

- Assume additional constants and primitive operators:
- hd : $\forall\alpha. \alpha$ list -> $\alpha$
- tl: $\forall\alpha. \alpha$ list -> $\alpha$ list
- is_empty : $\forall\alpha. \alpha$ list -> bool
- (::) : $\forall\alpha. \alpha$ -> $\alpha$ list -> $\alpha$ list
- [] : $\forall\alpha. \alpha$ list

## Polymorphic Example

- Show:

?

---

{} |- let rec length =
fun l -> if is_empty l then 0
else 1 + length (tl l)
in length (2 :: []) + length(true :: []) : int



## Polymorphic Example: Let Rec Rule

- Show: (1) (2)

(1) {length:α list -> int} |- fun l -> … : α list -> int

(2) {length:∀α. α list -> int} |- length (2 :: []) + length(true :: []) : int

---

{} |- let rec length =
fun l -> if is_empty l then 0
else 1 + length (tl l)
in length (2 :: []) + length(true :: []) : int



## Polymorphic Example (1)

- Show:

?

---

{length:α list -> int} |-
fun l -> if is_empty l then 0
else 1 + length (tl l)
: α list -> int



## Polymorphic Example (1): Fun Rule

- Show: (3)

---

{length:α list -> int, l: α list } |-
if is_empty l then 0
else length (hd l) + length (tl l) : int

---

{length:α list -> int} |-
fun l -> if is_empty l then 0
else 1 + length (tl l)
: α list -> int



## Polymorphic Example (3)

- Let Γ ={length:α list -> int, l: α list }
- Show

?

---

Γ|- if is_empty l then 0
else 1 + length (tl l) : int



## Polymorphic Example (3):IfThenElse

- Let Γ ={length:α list -> int, l: α list }
- Show

| (4) | (5) | (6) |
|---|---|---|
| Γ\|- is_empty l : bool | Γ\|- 0:int | Γ\|- 1 + length (tl l) : int |

---

Γ|- if is_empty l then 0
else 1 + length (tl l) : int

## Polymorphic Example (4)

- Let Γ ={length:α list -> int, l: α list }
- Show

?
———————————————
Γ|- is_empty l : bool



## Polymorphic Example (4):Application

- Let Γ ={length:α list -> int, l: α list }
- Show

? ?
——————————— ———————
Γ|- is_empty : α list -> bool Γ|- l : α list
———————————————
Γ|- is_empty l : bool



## Polymorphic Example (4)

- Let Γ ={length:α list -> int, l: α list }
- Show

By Const since α list -> bool
is instance {α→α} of
∀α. α list -> bool ?
——————————— ———————
Γ|- is_empty : α list -> bool Γ|- l : α list
———————————————
Γ|- is_empty l : bool



## Polymorphic Example (4)

- Let Γ ={length:α list -> int, l: α list }
- Show

By Const since α list -> bool is instance of ∀α. α list -> bool By Variable Γ(l) = α list
——————————— ———————
Γ|- is_empty : α list -> bool Γ|- l : α list
———————————————
Γ|- is_empty l : bool

- This finishes (4)



## Polymorphic Example (5):Const

- Let Γ ={length:α list -> int, l: α list }
- Show

By Const Rule

———————
Γ|- 0:int



## Polymorphic Example (6): BinOp

- Let Γ ={length:α list -> int, l: α list }
- Show

By Variable
———————
Γ|- length (7)
By Const : α list -> int ———————
——————— Γ|- (tl l) : α list
Γ|- 1:int App ———————————
Γ|- length (tl l) : int
———————————————
Γ|- 1 + length (tl l) : int

## Polymorphic Example (7):App Rule

- Let $\Gamma$ ={length:$\alpha$ list -> int, l: $\alpha$ list }
- Show

$$\dfrac{\dfrac{\text{Const}}{\Gamma \text{|- tl : } \alpha \text{ list -> } \alpha \text{ list}} \qquad \dfrac{\text{Variable}}{\Gamma \text{|- l : } \alpha \text{ list}}}{\Gamma \text{|- (tl l) : } \alpha \text{ list}}$$

By Const since $\alpha$ list -> $\alpha$ list is instance {$\alpha \rightarrow \alpha$} of $\forall\alpha.$ $\alpha$ list -> $\alpha$ list



## Polymorphic Example: (2) by BinOp

- Let $\Gamma'$ = {length:$\forall\alpha.$ $\alpha$ list -> int}
- Show:

$$\dfrac{\dfrac{(8)}{\Gamma' \text{ |- length (2 :: []) :int}} \qquad \dfrac{(9)}{\Gamma' \text{ |- length(true :: []) : int}}}{\text{\{length:}\forall\alpha.\ \alpha \text{ list -> int\} |- length (2 :: []) + length(true :: []) : int}}$$



## Polymorphic Example: (8)AppRule

- Let $\Gamma'$ = {length:$\forall\alpha.$ $\alpha$ list -> int}
- Show:

$$\dfrac{\dfrac{?}{\Gamma' \text{ |- length : int list ->int}} \qquad \dfrac{?}{\Gamma' \text{ |- (2 :: []):int list}}}{\Gamma' \text{ |- length (2 :: []) :int}}$$



## Polymorphic Example: (8)AppRule

- Let $\Gamma'$ = {length:$\forall\alpha.$ $\alpha$ list -> int}
- Show:

$$\dfrac{\dfrac{?}{\Gamma' \text{ |- length : int list ->int}} \qquad \dfrac{?}{\Gamma' \text{ |- (2 :: []):int list}}}{\Gamma' \text{ |- length (2 :: []) :int}}$$



## Polymorphic Example: (8)AppRule

- Let $\Gamma'$ = {length:$\forall\alpha.$ $\alpha$ list -> int}
- Show:

By Var since int list -> int is instance {$\alpha \rightarrow$ int} of $\forall\alpha.$ $\alpha$ list -> int (by $\alpha \rightarrow$ int)

$$\dfrac{\dfrac{}{\Gamma' \text{ |- length : int list ->int}} \qquad \dfrac{(10)}{\Gamma' \text{ |- (2 :: []):int list}}}{\Gamma' \text{ |- length (2 :: []) :int}}$$



## Polymorphic Example: (10)BinOpRule

- Let $\Gamma'$ = {length:$\forall\alpha.$ $\alpha$ list -> int}
- Show:

$$\dfrac{\dfrac{\text{Const}}{\Gamma' \text{ |- 2 : int}} \qquad \dfrac{?}{\Gamma' \text{ |- [] : int list}}}{\Gamma' \text{ |- (2 :: []) : int list}}$$

## Polymorphic Example: (10)BinOpRule

- Let $\Gamma'$ = {length:∀α. α list -> int}
- Show:
- By Const since int list is instance of ∀α. α list (by α ➔ int)

$$\frac{\overline{\Gamma' \mid\!- 2 : int} \quad \overline{\Gamma' \mid\!- [] : int\ list}}{\Gamma' \mid\!- (2 :: []) : int\ list}$$



## Polymorphic Example: (9)AppRule

- Let $\Gamma'$ = {length∀ α. α list -> int}
- Show:

$$\frac{\dfrac{?}{\Gamma' \mid\!- length : bool\ list \to int} \quad \dfrac{?}{\Gamma' \mid\!- (true :: []) : bool\ list}}{\Gamma' \mid\!- length\ (true :: []) : int}$$



## Polymorphic Example: (9)AppRule

- Let $\Gamma'$ = {length∀ α. α list -> int}
- Show:
- Var since bool list -> int is instance of ∀α. α list -> int (by α ➔ bool)

$$\frac{\overline{\Gamma' \mid\!- length : bool\ list \to int} \quad \dfrac{(10)}{\Gamma' \mid\!- (true :: []) : bool\ list}}{\Gamma' \mid\!- length\ (true :: []) : int}$$



## Polymorphic Example: (10)BinOpRule

- Let $\Gamma'$ = {length:∀α. α list -> int}
- Show:

$$\frac{\dfrac{Const}{\Gamma' \mid\!- true : bool} \quad \dfrac{?}{\Gamma' \mid\!- [] : bool\ list}}{\Gamma' \mid\!- (true :: []) : bool\ list}$$



## Polymorphic Example: (10)BinOpRule

- Let $\Gamma'$ = {length:∀α. α list -> int}
- Show:
- By Const since bool list is instance of ∀α. α list (by α ➔ bool)

$$\frac{\overline{\Gamma' \mid\!- true : bool} \quad \overline{\Gamma' \mid\!- [] : bool\ list}}{\Gamma' \mid\!- (true :: []) : bool\ list}$$



## Two Problems

- Type checking
  - Question: Does exp. e have type $\tau$ in env $\Gamma$?
  - Answer: Yes / No
  - Method: Type derivation
- Typability
  - Question Does exp. e have some type in env. $\Gamma$? If so, what is it?
  - Answer: Type $\tau$ / error
  - Method: Type inference

## Type Inference - Outline

- Begin by assigning a type variable as the type of the whole expression
- Decompose the expression into component expressions
- Use typing rules to generate constraints on components and whole
- Recursively find substitution that solves typing judgment of first subcomponent
- Apply substitution to next subcomponent and find substitution solving it; compose with first, etc.
- Apply comp of all substitution to orig. type var. to get answer



## Type Inference - Example

- What type can we give to

  (fun x -> fun f -> f (f x))
- Start with a type variable and then look at the way the term is constructed



## Type Inference - Example

- First approximate:

  $\{\ \}$ |- (fun x -> fun f -> f (f x)) : $\alpha$

- Second approximate: use fun rule

$$\frac{\{x : \beta\} \text{ |- (fun f -> f (f x))} : \gamma}{\{\ \} \text{ |- (fun x -> fun f -> f (f x))} : \alpha}$$

- Remember constraint $\alpha \equiv (\beta \rightarrow \gamma)$



## Type Inference - Example

- Third approximate: use fun rule

$$\frac{\dfrac{\{f : \delta\ ;\ x : \beta\} \text{ |- f (f x)} : \varepsilon}{\{x : \beta\} \text{ |- (fun f -> f (f x))} : \gamma}}{\{\ \} \text{ |- (fun x -> fun f -> f(f x))} : \alpha}$$

- $\alpha \equiv (\beta \rightarrow \gamma)$; $\gamma \equiv (\delta \rightarrow \varepsilon)$



## Type Inference - Example

- Fourth approximate: use app rule

$$\frac{\dfrac{\dfrac{\{f{:}\delta;\ x{:}\beta\}\text{|- f} : \varphi \rightarrow \varepsilon \quad \{f{:}\delta;\ x{:}\beta\}\text{|- f x} : \varphi}{\{f : \delta\ ;\ x : \beta\} \text{ |- (f (f x))} : \varepsilon}}{\{x : \beta\} \text{ |- (fun f -> f (f x))} : \gamma}}{\{\ \} \text{ |- (fun x -> fun f -> f (f x))} : \alpha}$$

- $\alpha \equiv (\beta \rightarrow \gamma)$; $\gamma \equiv (\delta \rightarrow \varepsilon)$



## Type Inference - Example

- Fifth approximate: use var rule, get constraint $\delta \equiv \varphi \rightarrow \varepsilon$, Solve with same
- Apply to next sub-proof

$$\frac{\dfrac{\dfrac{\{f{:}\delta;\ x{:}\beta\}\text{|- f} : \varphi \rightarrow \varepsilon \quad \{f{:}\delta;\ x{:}\beta\}\text{|- f x} : \varphi}{\{f : \delta\ ;\ x : \beta\} \text{ |- (f (f x))} : \varepsilon}}{\{x : \beta\} \text{ |- (fun f -> f (f x))} : \gamma}}{\{\ \} \text{ |- (fun x -> fun f -> f (f x))} : \alpha}$$

- $\alpha \equiv (\beta \rightarrow \gamma)$; $\gamma \equiv (\delta \rightarrow \varepsilon)$

## Type Inference - Example

- Current subst: $\{\delta \equiv \varphi \to \varepsilon\}$

$$\dfrac{\dfrac{\dfrac{\ldots \quad \{f:\varphi \to \varepsilon;\ x:\beta\} \mid\!- f\ x : \varphi}{\{f : \delta\ ;\ x : \beta\} \mid\!- (f\ (f\ x)) : \varepsilon}}{\{x : \beta\} \mid\!- (\text{fun } f \rightarrow f\ (f\ x)) : \gamma}}{\{\ \} \mid\!- (\text{fun } x \rightarrow \text{fun } f \rightarrow f\ (f\ x)) : \alpha}$$

- $\alpha \equiv (\beta \to \gamma);\ \gamma \equiv (\delta \to \varepsilon)$



## Type Inference - Example

- Current subst: $\{\delta \equiv \varphi \to \varepsilon\}$ Use App Rule

$$\dfrac{\dfrac{\dfrac{\dfrac{\{f:\varphi\to\varepsilon;\ x:\beta\} \mid\!- f:\zeta\to\varphi \quad \{f:\varphi\to\varepsilon;\ x:\beta\} \mid\!- x:\zeta}{\ldots \quad \{f:\varphi \to \varepsilon;\ x:\beta\} \mid\!- f\ x : \varphi}}{\{f : \delta\ ;\ x : \beta\} \mid\!- (f\ (f\ x)) : \varepsilon}}{\{x : \beta\} \mid\!- (\text{fun } f \rightarrow f\ (f\ x)) : \gamma}}{\{\ \} \mid\!- (\text{fun } x \rightarrow \text{fun } f \rightarrow f\ (f\ x)) : \alpha}$$

- $\alpha \equiv (\beta \to \gamma);\ \gamma \equiv (\delta \to \varepsilon)$



## Type Inference - Example

- Current subst: $\{\delta \equiv \varphi \to \varepsilon\}$
- Var rule: Solve $\zeta\to\varphi \equiv \varphi\to\varepsilon$ Unification

$$\dfrac{\dfrac{\dfrac{\dfrac{\{f:\varphi\to\varepsilon;\ x:\beta\} \mid\!- f:\zeta\to\varphi \quad \{f:\varphi\to\varepsilon;\ x:\beta\} \mid\!- x:\zeta}{\ldots \quad \{f:\varphi \to \varepsilon;\ x:\beta\} \mid\!- f\ x : \varphi}}{\{f : \delta\ ;\ x : \beta\} \mid\!- (f\ (f\ x)) : \varepsilon}}{\{x : \beta\} \mid\!- (\text{fun } f \rightarrow f\ (f\ x)) : \gamma}}{\{\ \} \mid\!- (\text{fun } x \rightarrow \text{fun } f \rightarrow f\ (f\ x)) : \alpha}$$

- $\alpha \equiv (\beta \to \gamma);\ \gamma \equiv (\delta \to \varepsilon)$



## Type Inference - Example

- Current subst: $\{\zeta \equiv \varepsilon,\ \varphi \equiv \varepsilon\} \circ \{\delta \equiv \varphi \to \varepsilon\}$
- Var rule: Solve $\zeta\to\varphi \equiv \varphi\to\varepsilon$ Unification

$$\dfrac{\dfrac{\dfrac{\dfrac{\{f:\varphi\to\varepsilon;\ x:\beta\} \mid\!- f:\zeta\to\varphi \quad \{f:\varphi\to\varepsilon;\ x:\beta\} \mid\!- x:\zeta}{\ldots \quad \{f:\varphi \to \varepsilon;\ x:\beta\} \mid\!- f\ x : \varphi}}{\{f : \delta\ ;\ x : \beta\} \mid\!- (f\ (f\ x)) : \varepsilon}}{\{x : \beta\} \mid\!- (\text{fun } f \rightarrow f\ (f\ x)) : \gamma}}{\{\ \} \mid\!- (\text{fun } x \rightarrow \text{fun } f \rightarrow f\ (f\ x)) : \alpha}$$

- $\alpha \equiv (\beta \to \gamma);\ \gamma \equiv (\delta \to \varepsilon)$



## Type Inference - Example

- Current subst: $\{\zeta \equiv \varepsilon,\ \varphi \equiv \varepsilon,\ \delta \equiv \varepsilon\to\varepsilon\}$
- Apply to next sub-proof

$$\dfrac{\dfrac{\dfrac{\dfrac{\ldots \quad \{f:\varepsilon\to\varepsilon;\ x:\beta\} \mid\!- x:\varepsilon}{\ldots \quad \{f:\varphi \to \varepsilon;\ x:\beta\} \mid\!- f\ x : \varphi}}{\{f : \delta\ ;\ x : \beta\} \mid\!- (f\ (f\ x)) : \varepsilon}}{\{x : \beta\} \mid\!- (\text{fun } f \rightarrow f\ (f\ x)) : \gamma}}{\{\ \} \mid\!- (\text{fun } x \rightarrow \text{fun } f \rightarrow f\ (f\ x)) : \alpha}$$

- $\alpha \equiv (\beta \to \gamma);\ \gamma \equiv (\delta \to \varepsilon)$



## Type Inference - Example

- Current subst: $\{\zeta \equiv \varepsilon,\ \varphi \equiv \varepsilon,\ \delta \equiv \varepsilon\to\varepsilon\}$
- Var rule: $\varepsilon \equiv \beta$

$$\dfrac{\dfrac{\dfrac{\dfrac{\ldots \quad \dfrac{}{\{f:\varepsilon\to\varepsilon;\ x:\beta\} \mid\!- x:\varepsilon}}{\ldots \quad \{f:\varphi \to \varepsilon;\ x:\beta\} \mid\!- f\ x : \varphi}}{\{f : \delta\ ;\ x : \beta\} \mid\!- (f\ (f\ x)) : \varepsilon}}{\{x : \beta\} \mid\!- (\text{fun } f \rightarrow f\ (f\ x)) : \gamma}}{\{\ \} \mid\!- (\text{fun } x \rightarrow \text{fun } f \rightarrow f\ (f\ x)) : \alpha}$$

- $\alpha \equiv (\beta \to \gamma);\ \gamma \equiv (\delta \to \varepsilon)$

## Type Inference - Example

- Current subst: $\{\varepsilon \equiv \beta\} o \{\zeta \equiv \varepsilon,\ \varphi \equiv \varepsilon,\ \delta \equiv \varepsilon \to \varepsilon\}$
- Solves subproof; return one layer

$$\ldots \quad \{f:\varepsilon \to \varepsilon;\ x:\beta\} \mid\!-\ x:\varepsilon$$
$$\ldots \quad \{f:\varphi \to \varepsilon;\ x:\beta\} \mid\!-\ f\ x : \varphi$$
$$\{f : \delta\ ;\ x : \beta\} \mid\!-\ (f\ (f\ x)) : \varepsilon$$
$$\{x : \beta\} \mid\!-\ (\text{fun } f \texttt{->} f\ (f\ x)) : \gamma$$
$$\{\ \} \mid\!-\ (\text{fun } x \texttt{->} \text{fun } f \texttt{->} f\ (f\ x)) : \alpha$$

- $\alpha \equiv (\beta \to \gamma);\ \gamma \equiv (\delta \to \varepsilon)$



## Type Inference - Example

- Current subst: $\{\varepsilon \equiv \beta,\ \zeta \equiv \beta,\ \varphi \equiv \beta,\ \delta \equiv \beta \to \beta\}$
- Solves this subproof; return one layer

$$\ldots$$
$$\ldots \quad \{f:\varphi \to \varepsilon;\ x:\beta\} \mid\!-\ f\ x : \varphi$$
$$\{f : \delta\ ;\ x : \beta\} \mid\!-\ (f\ (f\ x)) : \varepsilon$$
$$\{x : \beta\} \mid\!-\ (\text{fun } f \texttt{->} f\ (f\ x)) : \gamma$$
$$\{\ \} \mid\!-\ (\text{fun } x \texttt{->} \text{fun } f \texttt{->} f\ (f\ x)) : \alpha$$

- $\alpha \equiv (\beta \to \gamma);\ \gamma \equiv (\delta \to \varepsilon)$



## Type Inference - Example

- Current subst: $\{\varepsilon \equiv \beta,\ \zeta \equiv \beta,\ \varphi \equiv \beta,\ \delta \equiv \beta \to \beta\}$
- Need to satisfy constraint $\gamma \equiv (\delta \to \varepsilon)$, given subst, becomes: $\gamma \equiv ((\beta \to \beta) \to \beta)$

$$\ldots$$
$$\{f : \delta\ ;\ x : \beta\} \mid\!-\ (f\ (f\ x)) : \varepsilon$$
$$\{x : \beta\} \mid\!-\ (\text{fun } f \texttt{->} f\ (f\ x)) : \gamma$$
$$\{\ \} \mid\!-\ (\text{fun } x \texttt{->} \text{fun } f \texttt{->} f\ (f\ x)) : \alpha$$

- $\alpha \equiv (\beta \to \gamma);\ \gamma \equiv (\delta \to \varepsilon)$



## Type Inference - Example

- Current subst:

$\{\gamma \equiv ((\beta \to \beta) \to \beta), \varepsilon \equiv \beta,\ \zeta \equiv \beta,\ \varphi \equiv \beta,\ \delta \equiv \beta \to \beta\}$

- Solves subproof; return one layer

$$\ldots$$
$$\{f : \delta\ ;\ x : \beta\} \mid\!-\ (f\ (f\ x)) : \varepsilon$$
$$\{x : \beta\} \mid\!-\ (\text{fun } f \texttt{->} f\ (f\ x)) : \gamma$$
$$\{\ \} \mid\!-\ (\text{fun } x \texttt{->} \text{fun } f \texttt{->} f\ (f\ x)) : \alpha$$

- $\alpha \equiv (\beta \to \gamma);\ \gamma \equiv (\delta \to \varepsilon)$



## Type Inference - Example

- Current subst:

$\{\gamma \equiv ((\beta \to \beta) \to \beta), \varepsilon \equiv \beta,\ \zeta \equiv \beta,\ \varphi \equiv \beta,\ \delta \equiv \beta \to \beta\}$

- Need to satisfy constraint $\alpha \equiv (\beta \to \gamma)$ given subst: $\alpha \equiv (\beta \to ((\beta \to \beta) \to \beta))$

$$\ldots$$
$$\{x : \beta\} \mid\!-\ (\text{fun } f \texttt{->} f\ (f\ x)) : \gamma$$
$$\{\ \} \mid\!-\ (\text{fun } x \texttt{->} \text{fun } f \texttt{->} f\ (f\ x)) : \alpha$$

- $\alpha \equiv (\beta \to \gamma);$



## Type Inference - Example

- Current subst:

$\{\alpha \equiv (\beta \to ((\beta \to \beta) \to \beta)),$
$\gamma \equiv ((\beta \to \beta) \to \beta), \varepsilon \equiv \beta,\ \zeta \equiv \beta,\ \varphi \equiv \beta,\ \delta \equiv \beta \to \beta\}$

- Solves subproof; return on layer

$$\{x : \beta\} \mid\!-\ (\text{fun } f \texttt{->} f\ (f\ x)) : \gamma$$
$$\{\ \} \mid\!-\ (\text{fun } x \texttt{->} \text{fun } f \texttt{->} f\ (f\ x)) : \alpha$$

## Type Inference - Example

- Current subst:

$\{\alpha \equiv (\beta \rightarrow ((\beta \rightarrow \beta) \rightarrow \beta)),$

$\gamma \equiv ((\beta \rightarrow \beta) \rightarrow \beta), \varepsilon \equiv \beta, \zeta \equiv \beta, \varphi \equiv \beta, \delta \equiv \beta \rightarrow \beta\}$

- Done: $\alpha \equiv (\beta \rightarrow ((\beta \rightarrow \beta) \rightarrow \beta))$

{ } |- (fun x -> fun f -> f (f x)) : $\alpha$