

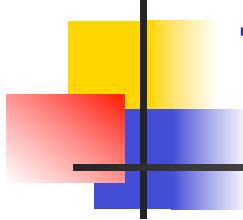
# Programming Languages and Compilers (CS 421)



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

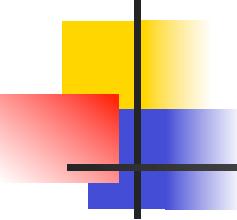


# Type Variables in Rules

- If\_then\_else rule:

$$\frac{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \tau \quad \Gamma \vdash e_3 : \tau}{\Gamma \vdash (\text{if } e_1 \text{ then } e_2 \text{ else } e_3) : \tau}$$

- $\tau$  is a type variable (meta-variable)
- Can take any type at all
- All instances in a rule application must get same type
- Then branch, else branch and if\_then\_else must all have same type



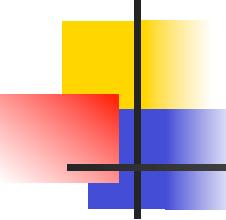
# Function Application

---

- Application rule:

$$\frac{\Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash (e_1 e_2) : \tau_2}$$

- If you have a function expression  $e_1$  of type  $\tau_1 \rightarrow \tau_2$  applied to an argument  $e_2$  of type  $\tau_1$ , the resulting expression  $e_1 e_2$  has type  $\tau_2$



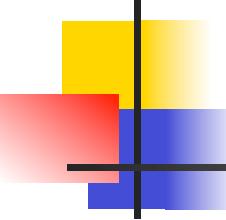
# Application Examples

$$\frac{\Gamma \vdash \text{print\_int} : \text{int} \rightarrow \text{unit} \quad \Gamma \vdash 5 : \text{int}}{\Gamma \vdash (\text{print\_int } 5) : \text{unit}}$$

- $e_1 = \text{print\_int}$ ,  $e_2 = 5$ ,
- $\tau_1 = \text{int}$ ,  $\tau_2 = \text{unit}$

$$\frac{\Gamma \vdash \text{map print\_int} : \text{int list} \rightarrow \text{unit list} \quad \Gamma \vdash [3;7] : \text{int list}}{\Gamma \vdash (\text{map print\_int } [3; 7]) : \text{unit list}}$$

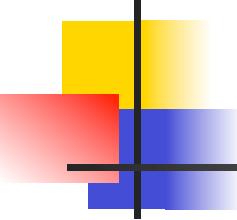
- $e_1 = \text{map print\_int}$ ,  $e_2 = [3; 7]$ ,
- $\tau_1 = \text{int list}$ ,  $\tau_2 = \text{unit list}$



# Fun Rule

- Rules describe types, but also how the environment  $\Gamma$  may change
- Can only do what rule allows!
- fun rule:

$$\frac{\{x : \tau_1\} + \Gamma \vdash e : \tau_2}{\Gamma \vdash \text{fun } x \rightarrow e : \tau_1 \rightarrow \tau_2}$$



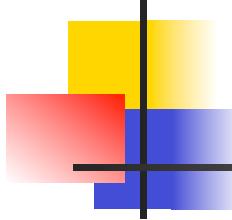
# Fun Examples

---

$$\frac{\{y : \text{int}\} + \Gamma \vdash y + 3 : \text{int}}{\Gamma \vdash \text{fun } y \rightarrow y + 3 : \text{int} \rightarrow \text{int}}$$

$$\underline{\{f : \text{int} \rightarrow \text{bool}\} + \Gamma \vdash f 2 :: [\text{true}] : \text{bool list}}$$

$$\begin{aligned} \Gamma \vdash & (\text{fun } f \rightarrow f 2 :: [\text{true}]) \\ & : (\text{int} \rightarrow \text{bool}) \rightarrow \text{bool list} \end{aligned}$$



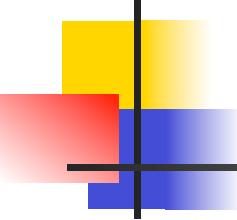
# (Monomorphic) Let and Let Rec

- let rule:

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \{x: \tau_1\} + \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (\text{let } x = e_1 \text{ in } e_2) : \tau_2}$$

- let rec rule:

$$\frac{\{x: \tau_1\} + \Gamma \vdash e_1 : \tau_1 \quad \{x: \tau_1\} + \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (\text{let rec } x = e_1 \text{ in } e_2) : \tau_2}$$



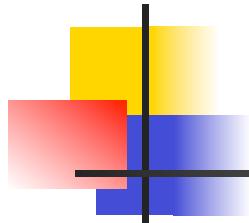
# Example

- Which rule do we apply?

?

---

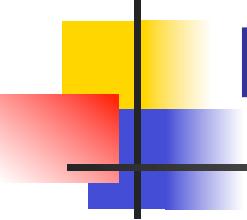
```
| - (let rec one = 1 :: one in  
    let x = 2 in  
        fun y -> (x :: y :: one) ) : int → int list
```



# Example

- Let rec rule: ② {one : int list} |-  
① {one : int list} |-  
(let x = 2 in  
fun y -> (x :: y :: one))  
(1 :: one) : int list : int → int list

```
| - (let rec one = 1 :: one in  
    let x = 2 in  
        fun y -> (x :: y :: one) ) : int → int list
```

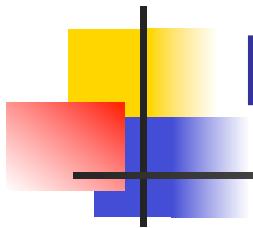


# Proof of 1

---

- Which rule?

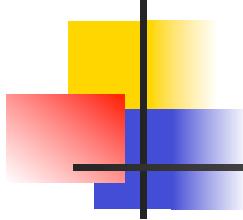
$$\{ \text{one} : \text{int list} \} \vdash (1 :: \text{one}) : \text{int list}$$



# Proof of 1

- Application

$$\frac{\begin{array}{c} \textcircled{3} \\ \{ \text{one} : \text{int list} \} \dashv \\ ((::) 1) : \text{int list} \rightarrow \text{int list} \end{array} \quad \begin{array}{c} \textcircled{4} \\ \{ \text{one} : \text{int list} \} \dashv \\ \text{one} : \text{int list} \end{array}}{\{ \text{one} : \text{int list} \} \dashv (1 :: \text{one}) : \text{int list}}$$



# Proof of 3

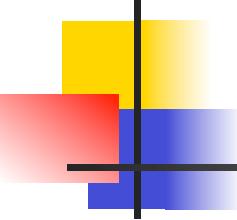
Constants Rule

Constants Rule

---

$$\frac{\{ \text{one} : \text{int list} \} \dashv \\ (\text{::}) : \text{int} \rightarrow \text{int list} \rightarrow \text{int list} \quad 1 : \text{int}}{\{ \text{one} : \text{int list} \} \dashv ((\text{::}) 1) : \text{int list} \rightarrow \text{int list}}$$

---



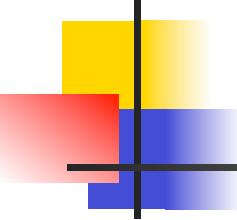
# Proof of 4

---

- Rule for variables

---

$$\overline{\{one : \text{int list}\} \vdash one : \text{int list}}$$



## Proof of 2

- Constant

⑤  $\{x:\text{int}; \text{one} : \text{int list}\} \vdash$   
fun y ->

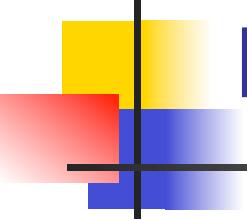
$(x :: y :: \text{one}))$

---

$\{\text{one} : \text{int list}\} \vdash 2:\text{int}$        $: \text{int} \rightarrow \text{int list}$

---

$\{\text{one} : \text{int list}\} \vdash (\text{let } x = 2 \text{ in}$   
fun y ->  $(x :: y :: \text{one})) : \text{int} \rightarrow \text{int list}$

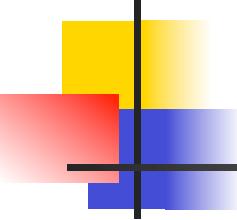


# Proof of 5

?

---

$$\frac{\{x:\text{int}; \text{one} : \text{int list}\} \vdash \text{fun } y \rightarrow (x :: y :: \text{one})}{\quad : \text{int} \rightarrow \text{int list}}$$



# Proof of 5

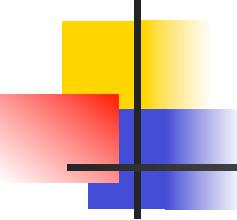
?

---

$$\{y:\text{int}; x:\text{int}; \text{one} : \text{int list}\} \vdash (x :: y :: \text{one}) : \text{int list}$$

---

$$\begin{aligned} & \{x:\text{int}; \text{one} : \text{int list}\} \vdash \text{fun } y \rightarrow (x :: y :: \text{one}) \\ & \quad : \text{int} \rightarrow \text{int list} \end{aligned}$$



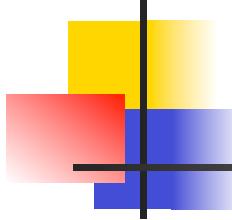
# Proof of 5

⑥

⑦

$$\frac{\{y:\text{int}; x:\text{int}; \text{one}:\text{int list}\} \quad \{y:\text{int}; x:\text{int}; \text{one}:\text{int list}\} \\ \vdash ((::) x):\text{int list} \rightarrow \text{int list} \quad \vdash (y :: \text{one}) : \text{int list}}{\{y:\text{int}; x:\text{int}; \text{one} : \text{int list}\} \vdash (x :: y :: \text{one}) : \text{int list}}$$

$$\frac{\{y:\text{int}; x:\text{int}; \text{one} : \text{int list}\} \vdash (x :: y :: \text{one}) : \text{int list}}{\{x:\text{int}; \text{one} : \text{int list}\} \vdash \text{fun } y \rightarrow (x :: y :: \text{one}) \\ : \text{int} \rightarrow \text{int list}}$$



# Proof of 6

Constant

Variable

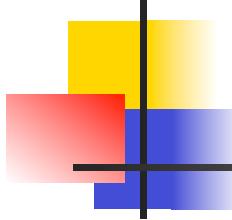
---

 $\{...\} \vdash (::)$ 

---

$$\frac{\{...\} \vdash (::) \quad : \text{int} \rightarrow \text{int list} \rightarrow \text{int list} \quad \{..., x:\text{int}, ...\} \vdash x:\text{int}}{\{y:\text{int}; x:\text{int}; \text{one} : \text{int list}\} \vdash ((::) x)}$$

:int list → int list



# Proof of 7

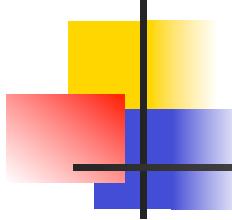
Pf of 6 [y/x]

⋮

$$\frac{\overline{\{y:\text{int}; \dots\} \vdash ((::) y)} \quad \overline{\{\dots; \text{one}: \text{int list}\} \vdash \text{one}: \text{int list}}}{\text{one}: \text{int list}}$$

$$\frac{}{\{y:\text{int}; x:\text{int}; \text{one} : \text{int list}\} \vdash (y :: \text{one}) : \text{int list}}$$

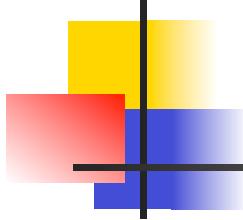
Variable



# Curry - Howard Isomorphism

---

- Type Systems are logics; logics are type systems
  - Types are propositions; propositions are types
  - Terms are proofs; proofs are terms
- 
- Functions space arrow corresponds to implication; application corresponds to modus ponens



# Curry - Howard Isomorphism

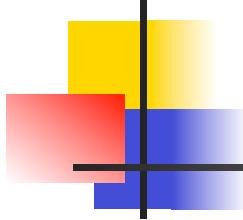
## ■ Modus Ponens

$$\frac{A \Rightarrow B \quad A}{B}$$

## • Application

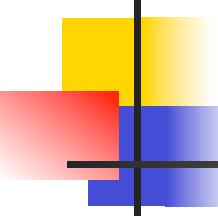
$$\frac{\Gamma \vdash e_1 : \alpha \rightarrow \beta \quad \Gamma \vdash e_2 : \alpha}{\Gamma \vdash (e_1 \ e_2) : \beta}$$

- The above system can't handle polymorphism as in OCAML
- No type variables in type language (only meta-variable in the logic)
- 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



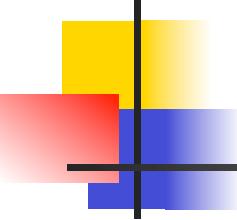
# Support for Polymorphic Types

- Monomorphic 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
    - $\forall \alpha_1, \dots, \alpha_n . \tau$
    - Can think of  $\tau$  as same as  $\forall. \tau$



# Support for Polymorphic Types

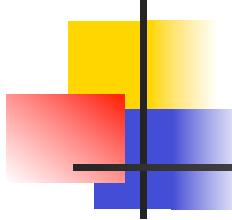
- Typing Environment  $\Gamma$  supplies polymorphic types (which will often just be monomorphic) for variables
- Free variables of monomorphic type just type variables that occur in it
  - Write  $\text{FreeVars}(\tau)$
- Free variables of polymorphic type removes variables that are universally quantified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



# 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  $\Gamma$
- $\text{Gen}(\tau, \Gamma) = \forall \alpha_1, \dots, \alpha_n . \tau$  where  
 $\{\alpha_1, \dots, \alpha_n\} = \text{freeVars}(\tau) - \text{freeVars}(\Gamma)$



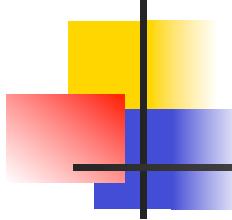
# Polymorphic Typing Rules

---

- A *type judgement* has the form

$$\Gamma \vdash \text{exp} : \tau$$

- $\Gamma$  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



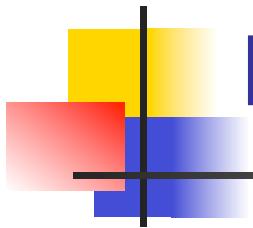
# Polymorphic Let and Let Rec

- let rule:

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

- let rec rule:

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



# Polymorphic Variables (Identifiers)

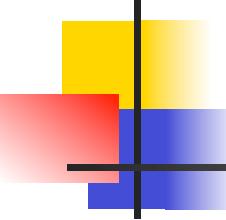
Variable axiom:

$$\frac{}{\Gamma \vdash x : \varphi(\tau)} \quad \text{if } \Gamma(x) = \forall \alpha_1, \dots, \alpha_n . \tau$$

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

$$\frac{}{\Gamma \vdash x : \tau} \quad \text{if } \Gamma(x) = \tau$$

- Constants treated same way

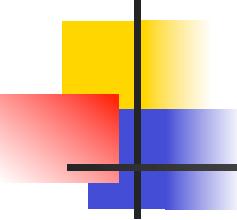


# Fun Rule Stays the Same

- fun rule:

$$\frac{\{x : \tau_1\} + \Gamma \vdash e : \tau_2}{\Gamma \vdash \text{fun } x \rightarrow e : \tau_1 \rightarrow \tau_2}$$

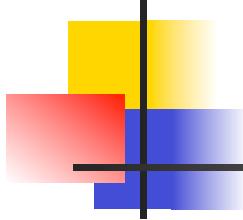
- Types  $\tau_1, \tau_2$  monomorphic
- Function argument must always be used at same type in function body



# Polymorphic Example

---

- Assume additional constants:
- $\text{hd} : \forall \alpha. \alpha \text{ list} \rightarrow \alpha$
- $\text{tl} : \forall \alpha. \alpha \text{ list} \rightarrow \alpha \text{ list}$
- $\text{is\_empty} : \forall \alpha. \alpha \text{ list} \rightarrow \text{bool}$
- $:: : \forall \alpha. \alpha \rightarrow \alpha \text{ list} \rightarrow \alpha \text{ list}$
- $[] : \forall \alpha. \alpha \text{ list}$



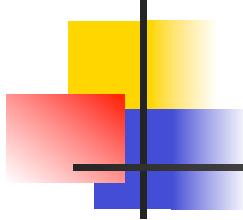
# Polymorphic Example

- 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)

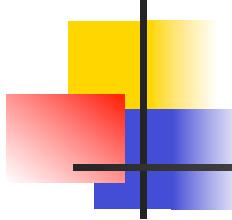
$$\{ \text{length}: \alpha \text{ list} \rightarrow \text{int} \}$$
$$\vdash \text{fun } l \rightarrow \dots$$
$$: \alpha \text{ list} \rightarrow \text{int}$$

(2)

$$\{ \text{length}: \forall \alpha. \alpha \text{ list} \rightarrow \text{int} \}$$
$$\vdash \text{length } ((::) 2 []) +$$
$$\text{length}((::) \text{true} []) : \text{int}$$

---

$$\{ \} \vdash \text{let rec length =}$$
$$\text{fun } l \rightarrow \text{if } \text{is\_empty } l \text{ then } 0$$
$$\text{else } 1 + \text{length } (\text{tl } l)$$
$$\text{in } \text{length } ((::) 2 []) + \text{length}((::) \text{true} []) : \text{int}$$



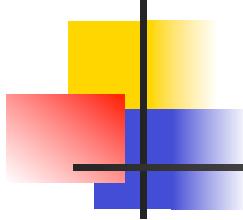
# Polymorphic Example (1)

- Show:

?

---

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



# Polymorphic Example (1): Fun Rule

- Show: (3)

$$\{ \text{length}: \alpha \text{ list} \rightarrow \text{int}, \quad l: \alpha \text{ list} \} \vdash$$

if `is_empty l` then 0

else `length (hd l) + length (tl l)` : int

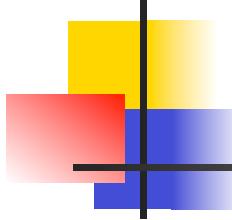
---

$$\{ \text{length}: \alpha \text{ list} \rightarrow \text{int} \} \vdash$$

fun `l` -> if `is_empty l` then 0

else 1 + `length (tl l)`

:  $\alpha$  list  $\rightarrow$  int



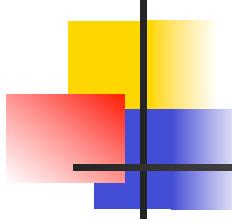
## Polymorphic Example (3)

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

?

---

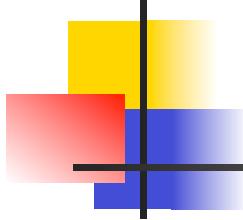
$$\begin{aligned} \Gamma |- & \text{ if } \text{is\_empty } \text{l} \text{ then } 0 \\ & \text{else } 1 + \text{length } (\text{tl } \text{l}) : \text{int} \end{aligned}$$



## Polymorphic Example (3):IfThenElse

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

$$\frac{\begin{array}{c} (4) \qquad \qquad \qquad (5) \qquad \qquad \qquad (6) \\ \Gamma \vdash \text{is\_empty } \text{l} \quad \Gamma \vdash 0:\text{int} \quad \Gamma \vdash 1 + \\ \qquad \qquad \qquad : \text{bool} \qquad \qquad \qquad \qquad \qquad \text{length } (\text{tl } \text{l}) : \text{int} \end{array}}{\Gamma \vdash \text{if is\_empty } \text{l} \text{ then } 0 \\ \qquad \qquad \qquad \text{else } 1 + \text{length } (\text{tl } \text{l}) : \text{int}}$$



## Polymorphic Example (4)

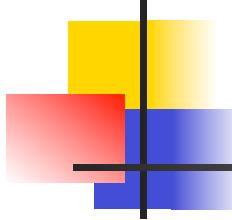
- Let  $\Gamma = \{\text{length} : \alpha \text{ list} \rightarrow \text{int}, \text{ l} : \alpha \text{ list}\}$
- Show

---

?

---

$\Gamma \vdash \text{is\_empty } \text{l} : \text{bool}$



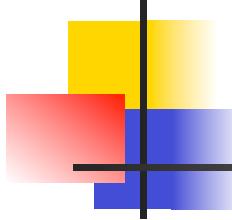
## Polymorphic Example (4): Application

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

?

?

$$\frac{\Gamma \vdash \text{is\_empty} : \alpha \text{ list} \rightarrow \text{bool} \quad \Gamma \vdash \text{l} : \alpha \text{ list}}{\Gamma \vdash \text{is\_empty l} : \text{bool}}$$

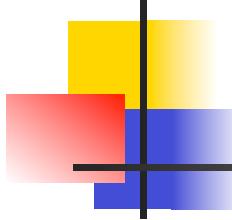


## Polymorphic Example (4)

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

By Const since  $\alpha \text{ list} \rightarrow \text{bool}$  is  
instance of  $\forall \alpha. \alpha \text{ list} \rightarrow \text{bool}$  ?

$$\frac{\overline{\Gamma \vdash \text{is\_empty} : \alpha \text{ list} \rightarrow \text{bool}} \quad \overline{\Gamma \vdash \text{l} : \alpha \text{ list}}}{\Gamma \vdash \text{is\_empty l} : \text{bool}}$$



## Polymorphic Example (4)

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

By Const since  $\alpha \text{ list} \rightarrow \text{bool}$  is By Variable  
instance of  $\forall \alpha. \alpha \text{ list} \rightarrow \text{bool}$   $\Gamma(\text{l}) = \alpha \text{ list}$

---

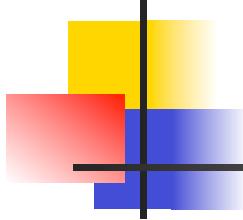
$$\Gamma \vdash \text{is\_empty} : \alpha \text{ list} \rightarrow \text{bool}$$

---

$$\Gamma \vdash \text{l} : \alpha \text{ list}$$

$$\Gamma \vdash \text{is\_empty l} : \text{bool}$$

- This finishes (4)



# Polymorphic Example (5):Const

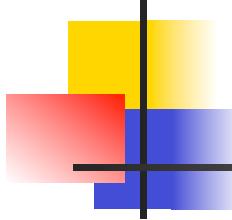
---

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

By Const Rule

---

$$\frac{}{\Gamma |- 0:\text{int}}$$



## Polymorphic Example (6):Arith Op

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

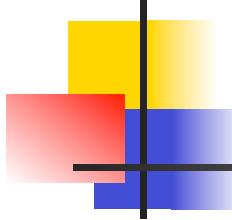
By Variable

$$\frac{}{\Gamma |- \text{length}} \quad (7)$$

By Const  $: \alpha \text{ list} \rightarrow \text{int} \quad \Gamma |- (\text{tl } \text{l}) : \alpha \text{ list}$

$$\frac{\Gamma |- 1 : \text{int}}{\Gamma |- \text{length} (\text{tl } \text{l}) : \text{int}}$$

$$\frac{}{\Gamma |- 1 + \text{length} (\text{tl } \text{l}) : \text{int}}$$



## Polymorphic Example (7):App Rule

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

By Const

By Variable

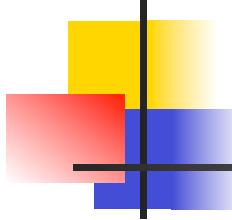
$$\frac{}{\Gamma \vdash \text{tl} : \alpha \text{ list} \rightarrow \alpha \text{ list}}$$

$$\frac{}{\Gamma \vdash \text{l} : \alpha \text{ list}}$$

$$\frac{}{\Gamma \vdash (\text{tl l}) : \alpha \text{ list}}$$

By Const since  $\alpha \text{ list} \rightarrow \alpha \text{ list}$  is instance of

$$\forall \alpha. \alpha \text{ list} \rightarrow \alpha \text{ list}$$



## Polymorphic Example: (2) by ArithOp

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

(8)

$\Gamma' \vdash$

$\text{length } ((::) 2 []) : \text{int}$

(9)

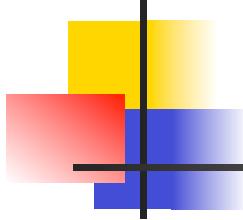
$\Gamma' \vdash$

$\text{length}((::) \text{true} []) : \text{int}$

---

$\{\text{length}: \alpha. \alpha \text{ list} \rightarrow \text{int}\}$

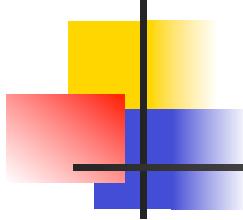
$\vdash \text{length } ((::) 2 []) + \text{length}((::) \text{true} []) : \text{int}$



## Polymorphic Example: (8)AppRule

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

$$\frac{\Gamma' \vdash \text{length} : \text{int list} \rightarrow \text{int} \quad \Gamma' \vdash ((::) 2 []): \text{int}}{\text{list}} \quad \Gamma' \vdash \text{length } ((::) 2 []): \text{int}$$



## Polymorphic Example: (8)AppRule

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

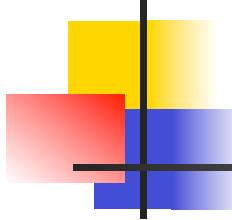
By Var since  $\text{int list} \rightarrow \text{int}$  is instance of

$\forall \alpha. \alpha \text{ list} \rightarrow \text{int}$

(10)

$$\frac{\Gamma' \vdash \text{length} : \text{int list} \rightarrow \text{int} \quad \Gamma' \vdash ((::) 2 []) : \text{int}}{\text{list}}$$

$$\Gamma' \vdash \text{length } ((::) 2 []) : \text{int}$$

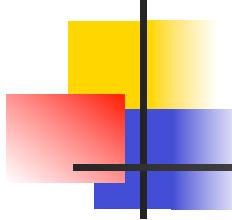


# Polymorphic Example: (10)AppRule

- Let  $\Gamma' = \{\text{length} \forall \alpha. \alpha \text{ list} \rightarrow \text{int}\}$
- Show:
- By Const since  $\alpha \text{ list}$  is instance of  
 $\forall \alpha. \alpha \text{ list}$

(11)

$$\frac{\Gamma' \vdash ((::) 2) : \text{int list} \rightarrow \text{int list} \quad \overline{\Gamma' \vdash [] : \text{int list}}}{\Gamma' \vdash ((::) 2 []) : \text{int list}}$$



# Polymorphic Example: (11)AppRule

- Let  $\Gamma' = \{\text{length} \forall \alpha. \alpha \text{ list} \rightarrow \text{int}\}$

- Show:

- By Const since  $\alpha$  list

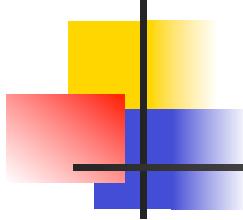
is instance of

$$\forall \alpha. \alpha \text{ list}$$
$$\frac{}{\Gamma' \vdash (\text{::}) : \text{int} \rightarrow \text{int list} \rightarrow \text{int list}}$$

int

By Const

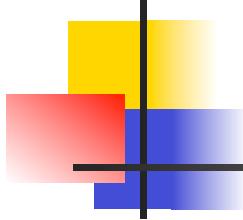
$$\frac{}{\Gamma' \vdash 2 :}$$
$$\Gamma' \vdash ((\text{::}) 2) : \text{int list} \rightarrow \text{int list}$$



# Polymorphic Example: (9)AppRule

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

$$\frac{\begin{array}{c} \Gamma' \vdash \\ \text{length:bool list} \rightarrow \text{int} \end{array} \quad \begin{array}{c} \Gamma' \vdash \\ ((::) \text{ true } []): \text{bool list} \end{array}}{\Gamma' \vdash \text{length } ((::) \text{ true } []): \text{int}}$$



# Polymorphic Example: (9)AppRule

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

By Var since  $\text{bool list} \rightarrow \text{int}$  is instance of

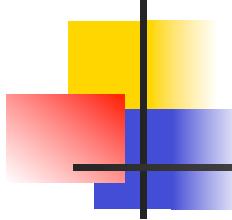
$\forall \alpha. \alpha \text{ list} \rightarrow \text{int}$

(12)

---

$$\frac{\Gamma' \vdash \text{length:bool list} \rightarrow \text{int} \quad \Gamma' \vdash ((::) \text{ true } []): \text{bool list}}{\Gamma' \vdash \text{length } ((::) \text{ true } []): \text{int}}$$

---



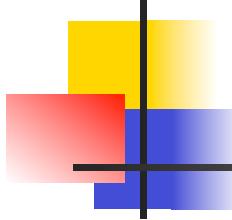
# Polymorphic Example: (12)AppRule

- Let  $\Gamma' = \{\text{length} \forall \alpha. \alpha \text{ list} \rightarrow \text{int}\}$
- Show:
- By Const since  $\alpha \text{ list}$  is instance of  
 $\forall \alpha. \alpha \text{ list}$

(13)

$$\frac{\Gamma' \vdash ((::)\text{true}): \text{bool list} \rightarrow \text{bool list} \quad \overline{\Gamma' \vdash []: \text{bool}}}{\text{list}}$$

$\Gamma' \vdash ((::)\text{true} []): \text{bool list}$



# Polymorphic Example: (13)AppRule

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

- Show:

By Const since bool list

is instance of  $\forall \alpha. \alpha \text{ list}$

By Const

---

$$\frac{}{\Gamma' \vdash -}$$
$$:::\text{bool} \rightarrow \text{bool list} \rightarrow \text{bool list}$$

---

$$\frac{}{\Gamma' \vdash -}$$
$$\text{true} : \text{bool}$$

---

$$\Gamma' \vdash ((::) \text{ true}) : \text{bool list} \rightarrow \text{bool list}$$