Programming Languages and Compilers (CS 421)

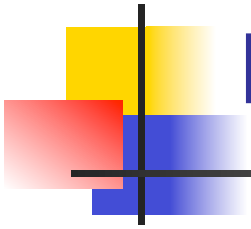
Elsa L Gunter

2112 SC, UIUC

http://www.cs.illinois.edu/class/cs421/

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

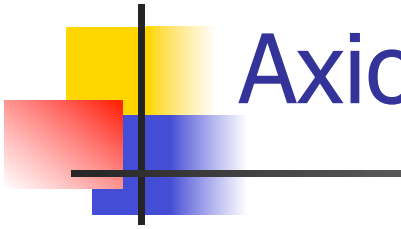
Format of Type Judgments



- A *type judgement* has the form

$$\Gamma \vdash exp : \tau$$

- Γ is a typing environment
  - Supplies the types of variables and functions
  - Γ is a list of the form [x : σ , . . .]
- exp is a program expression
- τ is a type to be assigned to exp
- |- pronounced "turnstyle", or "entails" (or "satisfies")

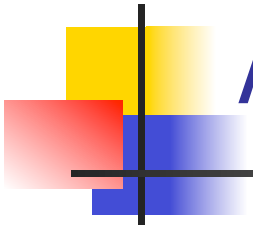


Axioms - Constants

$$\frac{}{\vdash n : \text{int}} \quad \text{(assuming } n \text{ is an integer constant)}$$

$$\frac{}{\vdash \text{true} : \text{bool}} \qquad \frac{}{\vdash \text{false} : \text{bool}}$$

- These rules are true with any typing environment
- *n* is a meta-variable



Axioms - Variables

Notation: Let $\Gamma(x) = \sigma$ if $x : \sigma \in \Gamma$ and there is no $x : \tau$ to the left of $x : \sigma$ in Γ

Variable axiom:

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

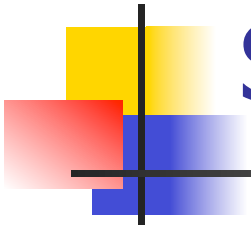
Simple Rules - Arithmetic

Primitive operators ($\oplus \in \{ +, -, *, \ldots \}$):

$$\frac{\Gamma \vdash e_1 : \text{int} \qquad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 \oplus e_2 : \text{int}}$$

Relations ($\sim \in \{ <, >, =, <=, >= \}$):

$$\frac{\Gamma \vdash e_1 : \text{int} \qquad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 \sim e_2 : \text{bool}}$$



Simple Rules - Booleans

Connectives

$$\frac{\Gamma \vdash e_1 : \text{bool} \qquad \Gamma \vdash e_2 : \text{bool}}{\Gamma \vdash e_1 \,\&\&\, e_2 : \text{bool}}$$

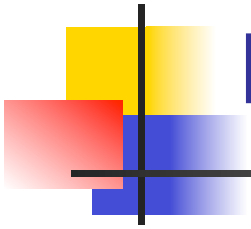
$$\frac{\Gamma \vdash e_1 : \text{bool} \qquad \Gamma \vdash e_2 : \text{bool}}{\Gamma \vdash e_1 \,||\, e_2 : \text{bool}}$$

Type Variables in Rules

- If_then_else rule:

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

- τ 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

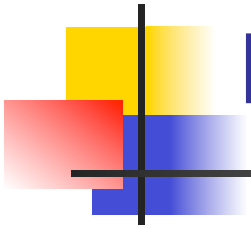


Function Application

- Application rule:

$$\frac{\Gamma \mid\!- e_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \mid\!- e_2 : \tau_1}{\Gamma \mid\!- (e_1\ e_2) : \tau_2}$$

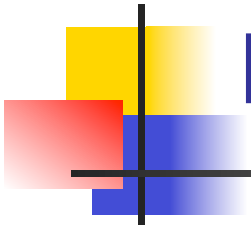
- If you have a function expression e_1 of type $\tau_1 \rightarrow \tau_2$ applied to an argument of type τ_1, the resulting expression has type τ_2



Fun Rule

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

$$\frac{[x : \tau_1] + \Gamma \text{ |- } e : \tau_2}{\Gamma \text{ |- fun } x \text{ -> } e : \tau_1 \rightarrow \tau_2}$$



Fun Examples

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

$$\frac{[f : int \rightarrow bool] + \Gamma \vdash f\ 2 :: [true] : bool\ list}{\Gamma \vdash (fun\ f \rightarrow f\ 2 :: [true]) : (int \rightarrow bool) \rightarrow bool\ list}$$

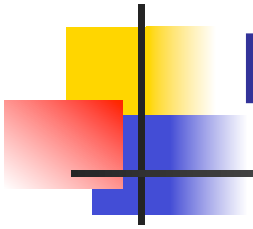
(Monomorphic) Let and Let Rec

- let rule:

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

- let rec rule:

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



Example

- Which rule do we apply?

$$\frac{?}{\begin{array}{l}\vdash (\text{let rec one} = 1 :: \text{one in} \\ \quad \text{let } x = 2 \text{ in} \\ \quad\quad \text{fun } y \text{ -> } (x :: y :: \text{one})) : \text{int} \rightarrow \text{int list}\end{array}}$$

Example

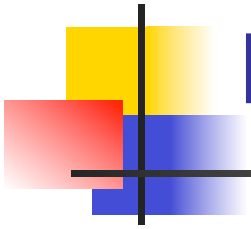
- Let rec rule:

① [one : int list] |- (1 :: one) : int list

② [one : int list] |- (let x = 2 in fun y -> (x :: y :: one)) : int → int list

---

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



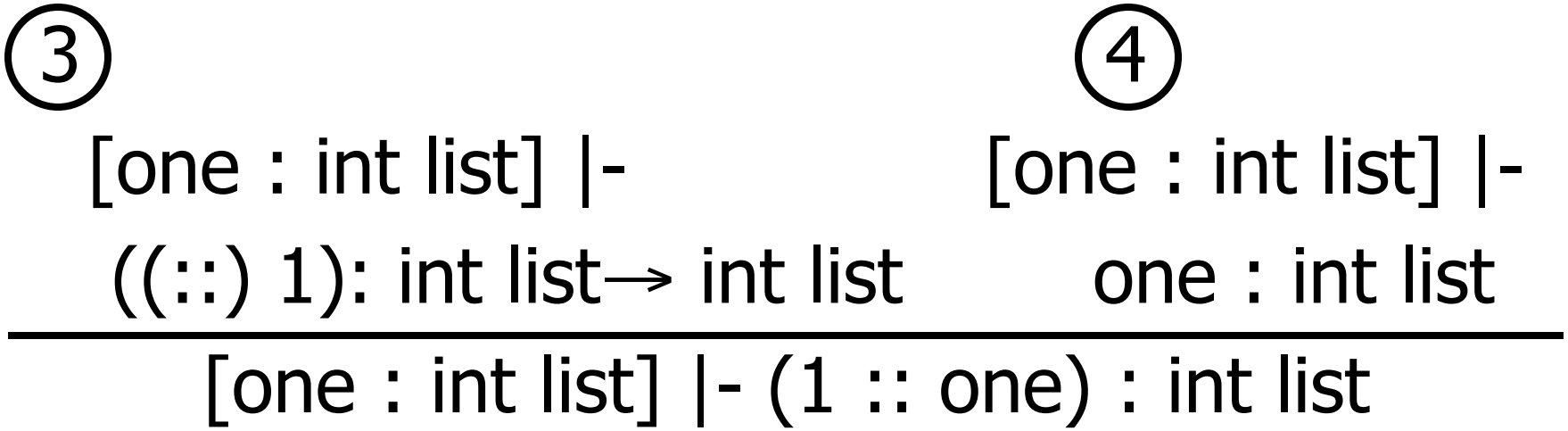
Proof of 1

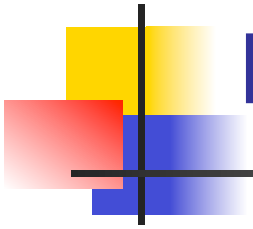
- Which rule?

[one : int list] |- (1 :: one) : int list

Proof of 1

- Application



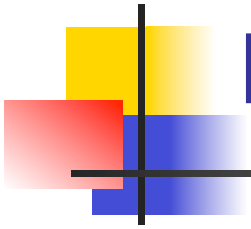


Proof of 3

Constants Rule Constants Rule

[one : int list] |- [one : int list] |-
(::) : int → int list→ int list 1 : int

[one : int list] |- ((::) 1) : int list → int list



Proof of 4

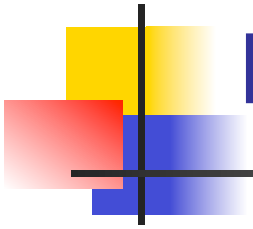
- Rule for variables

$$\frac{}{[one : int\ list] \vdash one : int\ list}$$

Proof of 2

- Constant

$$\frac{\dfrac{}{[\text{one} : \text{int list}] \vdash 2 : \text{int}} \quad ⑤\ [x{:}\text{int};\ \text{one} : \text{int list}] \vdash \text{fun y -> (x :: y :: one)} : \text{int} \to \text{int list}}{[\text{one} : \text{int list}] \vdash (\text{let x = 2 in fun y -> (x :: y :: one)}) : \text{int} \to \text{int list}}$$

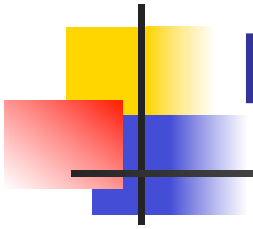


Proof of 5

?

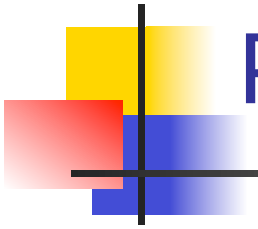
[x:int; one : int list] |- fun y -> (x :: y :: one))

: int → int list



Proof of 5

$$\dfrac{\dfrac{?}{[y{:}int;\ x{:}int;\ one : int\ list] \vdash (x :: y :: one) : int\ list}}{[x{:}int;\ one : int\ list] \vdash fun\ y \rightarrow (x :: y :: one)) : int \rightarrow int\ list}$$



Proof of 5

(6) [y:int; x:int; one : int list] |- ((::) x):int list→ int list

(7) [y:int; x:int; one : int list] |- (y :: one) : int list

[y:int; x:int; one : int list] |- (x :: y :: one) : int list

[x:int; one : int list] |- fun y -> (x :: y :: one))
: int → int list

Proof of 6

Constant

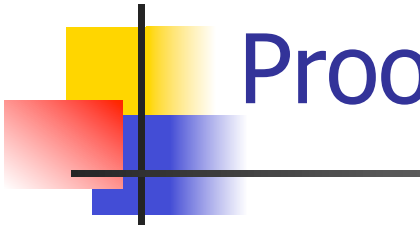
[…] |- (::)
: int→ int list→ int list

Variable

[…; x:int;…] |- x:int

[y:int; x:int; one : int list] |- ((::) x)
:int list→ int list

Proof of 7



Pf of 6 [y/x]

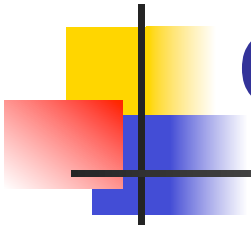
⋮

[y:int; …] |- ((::) y) :int list→ int list

Variable

[…; one: int list] |- one: int list

[y:int; x:int; one : int list] |- (y :: one) : int list



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

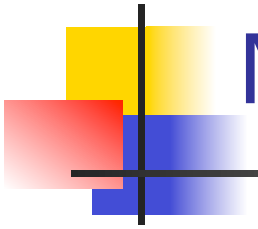
Curry - Howard Isomorphism

- Modus Ponens

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

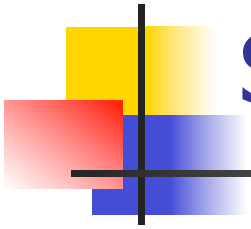
- Application

$$\frac{\Gamma \mid\!- e_1 : \alpha \rightarrow \beta \quad \Gamma \mid\!- e_2 : \alpha}{\Gamma \mid\!- (e_1\ e_2) : \beta}$$



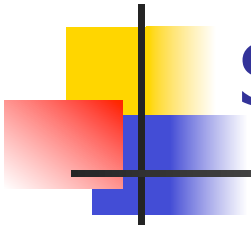
Mia Copa

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



Support for Polymorphic Types

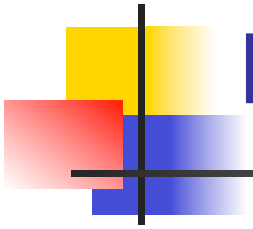
- Typing Environment Γ supplies polymorphic types (which will often just be monomorphic) for variables
- Free variables of monomorphic type just type variables that occur in it
  - Write FreeVars(τ)
- Free variables of polymorphic type removes variables that are universally quantified
  - FreeVars($\forall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

Monomorphic to Polymorphic

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

  $\{\alpha_1, \ldots, \alpha_n\}$ = freeVars(τ) – freeVars(Γ)

Polymorphic Typing Rules



- A *type judgement* has the form

  $$\Gamma \mid\!- \text{exp} : \tau$$

  - Γ uses polymorphic types
  - τ 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

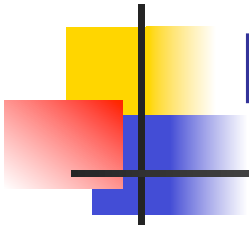
Polymorphic Let and Let Rec

- let rule:

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

- let rec rule:

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



Polymorphic Variables (Identifiers)

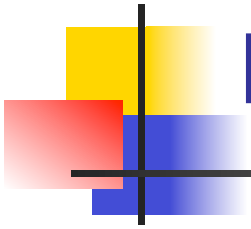
Variable axiom:

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

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

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

- Constants treated same way

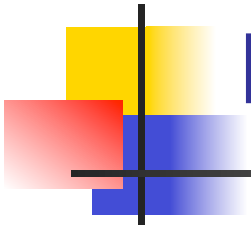


Fun Rule Stays the Same

- fun rule:

$$\frac{[x : \tau_1] + \Gamma \text{ |- } e : \tau_2}{\Gamma \text{ |- fun } x \text{ -> } e : \tau_1 \rightarrow \tau_2}$$

- Types τ_1, τ_2 monomorphic
- Function argument must always be used at same type in function body



Polymorphic Example

- Assume additional constants:
- hd :∀α. α list -> α
- tl: ∀α. α list -> α list
- is_empty : ∀α. α list -> bool
- :: : ∀α. α -> α list -> α list
- [] : ∀α. α list

Polymorphic Example

- Show:

$$\frac{\text{?}}{\begin{array}{l}\{\} \vdash \texttt{let rec length =} \\ \qquad \texttt{fun l -> if is\_empty l then 0} \\ \qquad\qquad \texttt{else 1 + length (tl l)} \\ \texttt{in length ((::) 2 []) + length((::) true []) : int}\end{array}}$$

Polymorphic Example: Let Rec Rule

- Show: (1) (2)

{length:α list -> int} {length:$\forall\alpha$. α list -> int}

|- fun l -> … |- length ((::) 2 []) +

: α list -> int 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:

$$\frac{\quad ? \quad}{\{\text{length}:\alpha\ \text{list} \rightarrow \text{int}\} \vdash \text{fun l -> if is\_empty l then 0 else 1 + length (tl l)} : \alpha\ \text{list} \rightarrow \text{int}}$$

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

$$\frac{?}{\Gamma \vdash \text{if is\_empty l then 0 else 1 + length (tl l)} : \text{int}}$$

Polymorphic Example (3):IfThenElse

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

$$\frac{\begin{array}{lll}(4) & (5) & (6)\\ \Gamma\text{|- is\_empty l} & \Gamma\text{|- 0:int} & \Gamma\text{|- 1 +}\\ \text{: bool} & & \text{length (tl l) : int}\end{array}}{\begin{array}{l}\Gamma\text{|- if is\_empty l then 0}\\ \text{else 1 + length (tl l) : int}\end{array}}$$

Polymorphic Example (4)

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

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

Polymorphic Example (4):Application

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

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

Polymorphic Example (4)

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

By Const since α list -> bool is instance of $\forall\alpha$. α list -> bool ?

$$\frac{\Gamma \vdash \text{is\_empty} : \alpha \text{ list} \to \text{bool} \qquad \Gamma \vdash \text{l} : \alpha \text{ list}}{\Gamma \vdash \text{is\_empty l} : \text{bool}}$$

Polymorphic Example (4)

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

By Const since α list -> bool is instance of $\forall\alpha$. α list -> bool

By Variable $\Gamma(l) = \alpha$ list

$$\frac{\Gamma \vdash \text{is\_empty} : \alpha \text{ list} \to \text{bool} \qquad \Gamma \vdash l : \alpha \text{ list}}{\Gamma \vdash \text{is\_empty } l : \text{bool}}$$

- This finishes (4)

Polymorphic Example (5):Const

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

By Const Rule

$$\frac{}{\Gamma \vdash 0:\text{int}}$$

Polymorphic Example (6):Arith Op

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

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash 1:\text{int}}\,\text{By Const} \qquad \dfrac{\dfrac{}{\Gamma \vdash \text{length} : \alpha \text{ list} \to \text{int}}\,\text{By Variable} \qquad \dfrac{}{\Gamma \vdash (\text{tl } l) : \alpha \text{ list}}\,(7)}{\Gamma \vdash \text{length (tl } l) : \text{int}}}{\Gamma \vdash 1 + \text{length (tl } l) : \text{int}}$$

Polymorphic Example (7):App Rule

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

$$\frac{\dfrac{}{\Gamma \vdash (\text{tl l}) : \alpha \text{ list} \rightarrow \alpha \text{ list}}\ \text{By Const} \qquad \dfrac{}{\Gamma \vdash \text{l} : \alpha \text{ list}}\ \text{By Variable}}{\Gamma \vdash (\text{tl l}) : \alpha \text{ list}}$$

By Const since α list -> α list is instance of

$\forall\alpha.\ \alpha$ list -> α list

Polymorphic Example: (2) by ArithOp

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

(8) (9)

Γ' |- Γ' |-

length ((::) 2 []) :int length((::) true []) : int

{length: α. α list -> int}

|- length ((::) 2 []) + length((::) true []) : int

Polymorphic Example: (8)AppRule

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

$$\frac{\Gamma' \text{ |- length : int list ->int} \quad \Gamma' \text{ |- ((::)2 []):int list}}{\Gamma' \text{ |- length ((::) 2 []) :int}}$$

Polymorphic Example: (8)AppRule

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

By Var since int list -> int is instance of

$\forall\alpha.\ \alpha$ list -> int

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

Polymorphic Example: (10)AppRule

- Let Γ' = {length:$\forall\alpha.\ \alpha$ list -> int}
- Show:
- By Const since α list is instance of

  $\forall\alpha.\ \alpha$ list

$$\frac{\Gamma' \text{|-((::) 2) : int list -> int list} \quad (11) \quad \Gamma' \text{ |- [] : int list}}{\Gamma' \text{ |- ((::) 2 []) :int list}}$$

Polymorphic Example: (11)AppRule

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

$$\dfrac{\dfrac{}{\Gamma' \vdash (::) : \text{int} \rightarrow \text{int list} \rightarrow \text{int list}} \quad \dfrac{\text{By Const}}{\Gamma' \vdash 2 : \text{int}}}{\Gamma' \vdash ((::)\ 2) : \text{int list} \rightarrow \text{int list}}$$

Polymorphic Example: (9)AppRule

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

$$\frac{\Gamma' \text{ |- length:bool list ->int} \qquad \Gamma' \text{ |- ((::) true []):bool list}}{\Gamma' \text{ |- length ((::) true []) :int}}$$

Polymorphic Example: (9)AppRule

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

By Var since bool list -> int is instance of

$\forall\alpha.\ \alpha$ list -> int

(12)

$$\frac{\Gamma' \vdash \text{length:bool list ->int} \qquad \Gamma' \vdash \text{((::) true []):bool list}}{\Gamma' \vdash \text{length ((::) true []) :int}}$$

Polymorphic Example: (12)AppRule

- Let Γ' = {length:$\forall\alpha.\ \alpha$ list -> int}
- Show:
- By Const since α list is instance of

$\forall\alpha.\ \alpha$ list

(13)

$$\frac{\Gamma' \text{|-((::)true):bool list ->bool list} \quad \overline{\Gamma' \text{|- []:bool list}}}{\Gamma' \text{ |- ((::) true []) :bool list}}$$

Polymorphic Example: (13)AppRule

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

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

By Const since bool list is instance of $\forall\alpha.\ \alpha$ list

By Const