Programming Languages and Compilers (CS 421)

Sasa Misailovic

4110 SC, UIUC

https://courses.engr.illinois.edu/cs421/fa2024/CS421C

Based on slides by Elsa Gunter, which are based in part on previous slides by Mattox Beckman and updated by Vikram Adve and Gul Agha

- Reminder from the last time

Inductive Proof System

- Hypotheses and Conclusion are logical formulas
- **Inference Rule:** Hypotheses imply Conclusion

$$\frac{\text{Hypothesis\_1} \quad \text{Hypothesis\_2} \quad \ldots \quad \text{Hypothesis\_n}}{\text{Conclusion}}$$

- **Axiom:** Holds without any previous hypothesis

$$\frac{}{\text{Conclusion}}$$

- Analogy: Axiom as a base case, Inference rule as an inductive case, Proof as a recursive derivation

Axioms – Constants (Monomorphic)

$$\frac{}{\Gamma \mid\!- \; n : \text{int}}$$ (assuming *n* is an integer constant)

$$\frac{}{\Gamma \mid\!- \; \text{true} : \text{bool}} \qquad \frac{}{\Gamma \mid\!- \; \text{false} : \text{bool}}$$

- These rules are true with any typing environment
- Γ, *n* are meta-variables

Axioms – Constants (Monomorphic)

$$\frac{}{\Gamma \ |\!- \ n : \text{int}}$$ (assuming *n* is an integer constant)

$$\frac{}{\Gamma \ |\!- \ \text{true} : \text{bool}} \qquad \frac{}{\Gamma \ |\!- \ \text{false} : \text{bool}}$$

- These rules are true with any typing environment
- Γ, *n* are meta-variables

Axioms – Variables (Monomorphic Rule)

Notation: Let $\Gamma(x) = \sigma$ if $x : \sigma \in \Gamma$

Note: if such σ exits, its unique

Variable axiom:

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

- The predicate $\Gamma(x) = \sigma$ is defined such that it is false if x has different type <u>or</u> x is not defined.

Simple Rules – Arithmetic (Mono)

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

$$\frac{\Gamma \mid\!- \; e_1 : \tau_1 \qquad \Gamma \mid\!- \; e_2 : \tau_2 \qquad (\oplus) : \tau_1 \to \tau_2 \to \tau_3}{\Gamma \mid\!- \; e_1 \oplus e_2 : \tau_3}$$

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

$$\frac{\Gamma \mid\!- \; e_1 : \tau \qquad \Gamma \mid\!- \; e_2 : \tau \qquad (\sim) : \tau \to \tau \to \text{bool}}{\Gamma \mid\!- \; e_1 \sim e_2 : \text{bool}}$$

All τ are **type variables**

For the moment, think τ is int or bool

Type Variables in Rules

- If_then_else rule:

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

- 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:

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

- If you have a function expression e_1 of type $\tau_1 \rightarrow \tau_2$ applied to an argument e_2 of type τ_1, the resulting expression $e_1 e_2$ has type τ_2

Example: Application

Function Application

- Application rule:

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

- Γ = {x:int, int_of_float:float -> int, y:float}

$$\frac{\Gamma \mid\!- \texttt{(fun z -> z > 3)} : \texttt{int -> bool} \qquad \Gamma \mid\!- \texttt{x} : \texttt{int}}{\Gamma \mid\!- \texttt{(fun z -> z > 3) x} : \texttt{bool}}$$

Fun Rule

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

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

(Monomorphic) Let and Let Rec

- let rule:

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

- let rec rule:

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

- Reminder ends

Limitations of the type system so far

- 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: $\alpha \to \beta$, 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 $\forall . \tau$

Example FreeVars Calculations

- Vars ('a -> (int -> 'b) -> 'a) =

- FreeVars (All 'b. 'a -> (int -> 'b) -> 'a) =

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

Example FreeVars Calculations

- Vars (‘a -> (int -> ‘b) -> ‘a) =

  {‘a , ‘b}

- FreeVars (All ‘b. ‘a -> (int -> ‘b) -> ‘a) =

- FreeVars {x : All ‘b. ‘a -> (int -> ‘b) -> ‘a,

  id: All ‘c. ‘c -> ‘c,

  y: All ‘c. ‘a -> ‘b -> ‘c} =

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} =

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

Polymorphic Typing Rules

- A *type judgement* has the form

$$\Gamma \text{ |- exp} : \tau$$

  - 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 Variables (Identifiers)

Variable axiom:

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

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

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

- Constants treated same way

Polymorphic Example (4)

- Let Γ ={lst: **int list** } assume is_empty is a special constant with type of $\forall\alpha.\ \alpha$ list -> bool

$$\frac{}{\Gamma \text{ |- is\_empty lst : } \mathbf{bool}}$$

Polymorphic Example (4)

- Let Γ ={lst: **int list** } assume is_empty is a special constant with type of $\forall\alpha.\ \alpha$ list -> bool

$$\frac{\Gamma \text{|- is\_empty : } \textbf{int list -> bool} \qquad \Gamma \text{|- lst } \textbf{: int list}}{\Gamma \text{|- is\_empty lst : } \textbf{bool}}$$

Polymorphic Example (4)

- Let Γ ={lst: **int list** } assume is_empty is a special constant with type of $\forall\alpha.\ \alpha$ list -> bool

By Variable

Γ(lst) = int list

$$\dfrac{\Gamma \text{ |- is\_empty : } \textbf{int list -> bool} \qquad \dfrac{\text{By Variable} \quad \Gamma(\text{lst}) = \text{int list}}{\Gamma \text{ |- lst : } \textbf{int list}}}{\Gamma \text{ |- is\_empty lst : } \textbf{bool}}$$

Constant ~~Variable~~ axiom:

$$\overline{\Gamma \text{ |- } x : \varphi(\tau)} \quad \sout{\text{if } \Gamma(x) = \forall\alpha_1, \ldots, \alpha_n . \tau}$$

- Where φ replaces all occurrences of $\alpha_1, \ldots, \alpha_n$ by monotypes $\tau_1, \ldots, \tau_n$

Polymorphic Example (4)

- Let Γ ={lst: **int list** } assume is_empty is a special constant with type of $\forall\alpha.\ \alpha$ list -> bool

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

By Variable Γ(lst) = int list

Γ|- is_empty : **int list -> bool** Γ|- lst **: int list**

Γ|- is_empty lst : **bool**

Constant ~~Variable~~ axiom:

$$\overline{\Gamma \ |\text{-}\ x : \varphi(\tau)}$$ ~~if $\Gamma(x) = \forall\alpha_1, \ldots, \alpha_n . \tau$~~

- Where φ replaces all occurrences of $\alpha_1, \ldots, \alpha_n$ by monotypes $\tau_1, \ldots, \tau_n$

Polymorphic Let and Let Rec

- let rule:

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

- let rec rule:

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

Polymorphic Let and Let Rec

- let rule:

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

- let rec rule:

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

Example

- let rule:

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

- let swap (x,y) = (y,x) in swap (1,2)
- swap should work for all types of x and y
- But we instantiate swap with concrete types

$$\frac{(1)\ \Gamma \mid\!- \; z\text{->}\textit{match z with (x,y) -> (y,x)} : \tau_1 \qquad (2)\ \{\textit{swap} : \mathrm{Gen}(\tau_1,\Gamma)\} + \Gamma \mid\!- \; \textit{swap (1,2)} : \tau_2}{\Gamma \mid\!- \; (\text{let swap} = e_1 \text{ in } e_2) : \tau_2}$$

Example

- let rule:

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

- let swap (x,y) = (y,x) in swap (1,2)
- swap should work for all types of x and y
- But we instantiate swap with concrete types

What is τ_1?

$$\frac{\Gamma \mid\!- \; z \text{->} \textit{match z with (x,y) -> (y,x)} : \tau_1 \quad \{\textit{swap} : \text{Gen}(\tau_1,\Gamma)\} + \Gamma \mid\!- \; \textit{swap (1,2)} : \tau_2}{\Gamma \mid\!- \; (\text{let swap} = e_1 \text{ in } e_2) : \tau_2}$$

Example

- let rule:

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

- let swap (x,y) = (y,x) in swap (1,2)
- swap should work for all types of x and y
- But we instantiate swap with concrete types

What is τ_1? ($\alpha * \beta \rightarrow \beta * \alpha$)

$$\frac{\Gamma \mid\!- z\text{->} \textit{match z with (x,y) -> (y,x)} : \tau_1 \quad \{\textit{swap} : \text{Gen}(\tau_1,\Gamma)\} + \Gamma \mid\!- \textit{swap (1,2)} : \tau_2}{\Gamma \mid\!- (\text{let swap} = e_1 \text{ in } e_2) : \tau_2}$$

Example

- let rule:

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

- let swap (x,y) = (y,x) in swap (1,2)
- swap should work for all types of x and y
- But we instantiate swap with concrete types

What is τ_1? ($\alpha*\beta\to\beta*\alpha$

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

$$\frac{\Gamma \;|\!-\; \ldots : \tau_1 \quad \{\mathit{swap} : \text{Gen}(\tau_1,\Gamma)\} + \Gamma \;|\!-\; \mathit{swap}\ (1,2) : \tau_2}{\Gamma \;|\!-\; (\text{let swap} = e_1 \text{ in } e_2) : \tau_2}$$

Example

- let rule:

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

- let swap (x,y) = (y,x) in swap (1,2)
- swap should work for all types of x and y
- But we instantiate swap with concrete types

What is τ_1? ($\alpha * \beta \rightarrow \beta * \alpha$

- $\mathrm{Gen}(\tau, \Gamma) = \forall \alpha_1, \ldots, \alpha_n . \tau$ where $\{\alpha_1, \ldots, \alpha_n\} = \mathrm{freeVars}(\tau) - \mathrm{freeVars}(\Gamma)$

$$\frac{\ldots \{swap : \forall \alpha,\beta.\ \alpha * \beta \rightarrow \beta * \alpha\} + \Gamma \mid\!- swap\ (1,2) : \tau_2}{\Gamma \mid\!- (\text{let swap} = e_1 \text{ in } e_2) : \tau_2}$$

Example

- 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 swap (x,y) = (y,x) in swap (1,2)
- swap should work for all types of x and y
- But we instantiate swap with concrete types

What is τ_2? (int*int→int*int)

$$\frac{\ldots \{swap : \forall\alpha,\beta.\ \alpha*\beta\rightarrow\beta*\alpha\} + \Gamma \mid\!- swap\ (1,2) : \tau_2}{\Gamma \mid\!- (\text{let swap} = e_1 \text{ in } e_2) : \tau_2}$$

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

We will discharge them all with the "Const" rule

Polymorphic Example

- Assume additional **constants and primitive operators**:
- (::) : $\forall\alpha.\ \alpha$ -> α list -> α list
- [] : $\forall\alpha.\ \alpha$ list

E.g., $\Gamma = \{\ \}$

$$\frac{\Gamma \mid\!- 1: \text{int} \quad \Gamma \mid\!- []: \text{int list} \quad \Gamma \mid\!- (::):\text{int} \to \text{int list} \to \text{int list}}{\Gamma \mid\!- 1::[]}$$

Polymorphic Example

- Assume additional **constants and primitive operators**:
- (::) : $\forall\alpha.\ \alpha$ -> α list -> α list
- [] : $\forall\alpha.\ \alpha$ list

Substitute α with int

$$
\dfrac{\Gamma \mid\!\!- 1 : \text{int} \quad \dfrac{\text{Const (instance of of } \forall\alpha.\ \alpha \text{ list)}}{\Gamma \mid\!\!- [\,] : \text{int list}} \quad \dfrac{\text{Const (instance of } \forall\ \alpha \text{ -> } \alpha \text{ list -> } \alpha \text{ list)}}{\Gamma \mid\!\!- (::) : \text{int->int list->int list}}}{\Gamma \mid\!\!- 1::[\,]}
$$

Polymorphic Example

- let rec rule:

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

- Show:

?

{} |- let rec length =

fun lst -> if is_empty lst then 0

else 1 + length (tl lst)

in

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

Polymorphic Example: Let Rec Rule (Repeat)

- Show: (1) (2)

(1)

{length:α list -> int}
|- fun lst -> …
: **α list -> int**

(2)

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

{} |- let rec length =

fun lst -> if is_empty lst then 0

else 1 + length (tl lst)

in

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

Polymorphic Example (1)

- fun rule:

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

- Show:

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

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

Polymorphic Example (1): Fun Rule

- fun rule:

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

- Show: (3)

{length:α list -> int, lst: α list } |-

if is_empty lst then 0

else length (hd lst) + length (tl lst) **: int**

{length:α list -> int} |-

fun lst -> if is_empty lst then 0

else 1 + length (tl lst)

: α list -> int

Polymorphic Example (3)

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

> - If_then_else rule:
>
> $$\frac{\Gamma \mid\!- e_1 : \text{bool} \quad \Gamma \mid\!- e_2 : \tau \quad \Gamma \mid\!- e_3 : \tau}{\Gamma \mid\!- (\text{if } e_1 \text{ then } e_2 \text{ else } e_3) : \tau}$$

$$\frac{?}{\Gamma \mid\!- \text{if is\_empty lst then 0 else 1 + length (tl lst)} \;\mathbf{: int}}$$

Polymorphic Example (3):IfThenElse

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

- If_then_else rule:

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

$$\frac{\overset{(4)}{\Gamma \mid\!- \text{ is\_empty lst } : \textbf{bool}} \qquad \overset{(5)}{\Gamma \mid\!- \text{ 0:int}} \qquad \overset{(6)}{\Gamma \mid\!- \text{ 1 + length (tl lst) } : \textbf{int}}}{\Gamma \mid\!- \text{ if is\_empty lst then 0 else 1 + length (tl lst) } : \textbf{int}}$$

Polymorphic Example (4)

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

$$\frac{?}{\Gamma \text{|- is\_empty lst : bool}}$$

Polymorphic Example (4):Application

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

$$\dfrac{\dfrac{?}{\Gamma \mid\!- \text{ is\_empty} : \alpha \text{ list} \to \text{bool}} \qquad \dfrac{?}{\Gamma \mid\!- \text{ lst} : \alpha \text{ list}}}{\Gamma \mid\!- \text{ is\_empty lst} : \text{bool}}$$

Polymorphic Example (4)

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

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

$$\dfrac{\dfrac{}{\Gamma \text{|- is\_empty} : \alpha\ \textbf{list -> bool}} \quad \dfrac{}{\Gamma \text{|- lst} : \alpha\ \textbf{list}}}{\Gamma \text{|- is\_empty lst} : \textbf{bool}}$$

Polymorphic Example (4)

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

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

By Variable Γ(lst) = α list

$$\dfrac{\dfrac{}{\Gamma \vdash \text{is\_empty} : \boldsymbol{\alpha\ \text{list} \rightarrow \text{bool}}} \qquad \dfrac{}{\Gamma \vdash \text{lst} : \boldsymbol{\alpha\ \text{list}}}}{\Gamma \vdash \text{is\_empty lst} : \textbf{bool}}$$

- This finishes (4)

Polymorphic Example (3):IfThenElse (Repeat)

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

(4) ✓ (5) (6)

Γ|- is_empty lst : **bool** $\quad$ Γ|- 0:int $\quad$ Γ|- 1 + length (tl lst) : **int**

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

Polymorphic Example (5):Const

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

By Const Rule

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

Polymorphic Example (3):IfThenElse (Repeat)

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

(4) ✓	(5) ✓	(6)
Γ|- is_empty lst **: bool**	Γ|- 0:int	Γ|- 1 + length (tl lst) **: int**

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

Polymorphic Example (6):Arith Op

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

$$\dfrac{\dfrac{\text{By Const}}{\Gamma |\text{- } 1 : \textbf{int}} \quad \dfrac{\dfrac{\text{By Variable}}{\Gamma |\text{- length} : \alpha \textbf{ list -> int}} \quad \dfrac{(7)}{\Gamma |\text{- (tl lst)} : \alpha \textbf{ list}}}{\Gamma |\text{- length (tl lst)} : \textbf{int}}}{\Gamma |\text{- } 1 + \text{length (tl lst)} : \textbf{int}}$$

Polymorphic Example (7):App Rule

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

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

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

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

Constant ~~Variable~~ axiom:

$$\overline{\Gamma\ |\text{-}\ x : \varphi(\tau)} \quad \text{~~if } \Gamma(x) = \forall\alpha_1, \dots, \alpha_n . \tau\text{~~}$$

- Where φ replaces all occurrences of $\alpha_1, \dots, \alpha_n$ by monotypes $\tau_1, \dots, \tau_n$

Polymorphic Example: Let Rec Rule (Repeat)

- Show: (1) ✓ (2)

$$\frac{\{length:\alpha\ list \rightarrow int\} \vdash \text{fun lst} \rightarrow \ldots : \alpha\ list \rightarrow int \qquad \{length: \forall \alpha.\ \alpha\ list \rightarrow int\} \vdash length\ (2 :: []) + length(true :: []) : int}{\{\} \vdash \text{let rec length = fun l -> if is\_empty l then 0 else 1 + length (tl l) in length (2 :: []) + length(true :: []) : int}}$$

{length:α list -> int} |- fun lst -> … : **α list -> int**

{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: (2) by ArithOp

- Let Γ' = {length:∀α. α list -> int}
- Show:

(8) Γ' |- length (2 :: []) :**int**

(9) Γ' |- length(true :: []) : **int**

{length: α. α list -> int}

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

Polymorphic Example: (8)AppRule

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

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

Polymorphic Example: (8)AppRule

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

By Var since int list -> int is instance of

$\forall\alpha.\ \alpha$ list -> int (by replacing α with int)

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

Variable axiom:

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

- Where φ replaces all occurrences of $\alpha_1, \ldots, \alpha_n$ by monotypes $\tau_1, \ldots, \tau_n$

Polymorphic Example: (10)AppRule

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

(11)

$$\frac{\Gamma' \mid\!\!- (2 ::) : \textbf{int list -> int list} \qquad \dfrac{}{\Gamma' \mid\!\!- [] : \textbf{int list}}}{\Gamma' \mid\!\!- (2 :: []) : \textbf{int list}}$$

Polymorphic Example: (11) AppRule

- Let Γ' = {length:∀α. α list -> int}
- Show:
- By Const since α list

  is instance of

$$\dfrac{\dfrac{\forall \alpha.\ \alpha\ \text{list}}{\Gamma' \ |\!-\ (::) : \textbf{int -> int list -> int list}} \qquad \dfrac{\text{By Const}}{\Gamma' \ |\!-\ 2 : \textbf{int}}}{\Gamma' \ |\!-\ (2\ ::\ ) : \textbf{int list -> int list}}$$

Polymorphic Example: (2) by ArithOp

- Let Γ' = {length:∀α. α list -> int}
- Show:

(8) ✓

Γ' |-

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

(9)

Γ' |-

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

{length: α. α list -> int}

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

Polymorphic Example: (9)AppRule

- Let Γ' = {length:∀α. α list -> int}
- Show:

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

Polymorphic Example: (9)AppRule

- Let Γ' = {length:∀α. α list -> int}
- Show:

By Var since bool list -> int is instance of

∀α. α list -> int

(12)

$$\dfrac{\dfrac{}{\Gamma' \text{ |- length : } \textbf{bool list ->int}} \qquad \dfrac{(12)}{\Gamma' \text{ |- (true :: []) :} \textbf{bool list}}}{\Gamma' \text{ |- length (true :: []) :} \textbf{int}}$$

Polymorphic Example: (12)AppRule

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

$\forall\alpha$. α list

(13)

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

Polymorphic Example: (13)AppRule

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

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

$$\dfrac{\dfrac{\text{By Const since bool list is instance of } \forall\alpha.\ \alpha \text{ list}}{\Gamma' \text{ |- } (::):\textbf{bool -> bool list -> bool list}} \quad \dfrac{\text{By Const}}{\Gamma' \text{ |- true : } \textbf{bool}}}{\Gamma' \text{ |- } ((::) \text{ true}) : \textbf{bool list -> bool list}}$$

Polymorphic Example: (2) by ArithOp

- Let Γ' = {length:∀α. α list -> int}
- Show:

(8) ✓

Γ' |-

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

(9) ✓

Γ' |-

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

{length: α. α list -> int}

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

Polymorphic Example: Let Rec Rule (Repeat)

- Show: (1) ✓ (2) ✓ .

{length:α list -> int} |- fun lst -> … : **α list -> int**

{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**