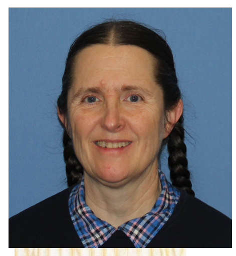
Programming Languages and Compilers (CS 421)

Elsa L Gunter

2112 SC, UIUC



https://courses.engr.illinois.edu/cs421/sp2023

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

Mea Culpa

- The above system can't handle polymorphism as in OCAML
- No type variables in type language (only meta-variables in the logic)
- Would need:
  - Object level type variables and some kind of type quantification
  - **let** and **let rec** rules to introduce polymorphism
  - Explicit changes to rules 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
  - $\text{FreeVars}(\forall\alpha_1, \ldots, \alpha_n . \tau) = \text{FreeVars}(\tau) - \{\alpha_1, \ldots, \alpha_n\}$
- FreeVars(Γ) = all FreeVars of types in range of Γ

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}

Monomorphic to Polymorphic

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

Polymorphic Typing Rules

- A *type judgement* has the form

$$\Gamma \mid\!- \exp : \tau$$

  - Γ uses polymorphic types
  - τ still monomorphic
- Most rules stay same (except use more general typing environments)
- Rules that change:
  - Variables, Constants
  - Primitive operators (monops and binops)
  - Let and Let Rec
- Worth noting functions again

Polymorphic Let and Let Rec

- let rule:

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

- let rec rule:

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

Fun Rule Stays the Same

- fun rule:

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

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

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, monops and binops treated similarly (with signatures)

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

Binary Operator Rule (Polymorphic)

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

Assume BinOp signature gives

$$\oplus : \forall \alpha_1, \ldots , \alpha_n . \tau_1 \rightarrow \tau_2 \rightarrow \tau_3$$

$$\frac{\Gamma \mid\!- e_1 : \tau'_1 \qquad \Gamma \mid\!- e_2 : \tau'_2}{\Gamma \mid\!- e_1 \oplus e_2 : \tau'_3} \{\alpha_1 \rightarrow \zeta_1, \ldots, \alpha_n \rightarrow \zeta_n\}$$

where τ'_i is τ_i with all α_i replaced by ζ_i

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)

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

Polymorphic Example (1)

- Show:

$$\frac{?}{\{length{:}\alpha\ list \to int\} \mid\!-\ fun\ l \to if\ is\_empty\ l\ then\ 0\ else\ 1 + length\ (tl\ l) : \alpha\ list \to int}$$

Polymorphic Example (1): Fun Rule

- Show: (3)

$$\frac{\{length:\alpha\ list \rightarrow int,\ \ l: \alpha\ list\ \} \mid\!- \text{if is\_empty l then 0 else length (hd l) + length (tl l) : int}}{\{length:\alpha\ list \rightarrow int\} \mid\!- \text{fun l -> if is\_empty l then 0 else 1 + length (tl l)} : \alpha\ list \rightarrow int}$$

Polymorphic Example (3)

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

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

Polymorphic Example (3):IfThenElse

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

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

Polymorphic Example (4)

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

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

Polymorphic Example (4):Application

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

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

Polymorphic Example (4)

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

By Const since α list -> bool

is instance {$\alpha \rightarrow \alpha$} of

$\forall\alpha.\ \alpha$ list -> bool ?

$$\frac{\dfrac{}{\Gamma \text{|- is\_empty : } \alpha \text{ list -> bool}} \qquad \dfrac{?}{\Gamma \text{|- l : } \alpha \text{ list}}}{\Gamma \text{|- is\_empty l : bool}}$$

Polymorphic Example (4)

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

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

By Variable Γ(l) = α list

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

- This finishes (4)

Polymorphic Example (5):Const

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

By Const Rule

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

Polymorphic Example (6): BinOp

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

$$
\dfrac{\dfrac{\text{By Const}}{\Gamma \mid\!-\ 1\text{:int}} \quad \dfrac{\dfrac{\text{By Variable}}{\Gamma \mid\!-\ \text{length} : \alpha\ \text{list -> int}} \quad \dfrac{(7)}{\Gamma \mid\!-\ (\text{tl l}) : \alpha\ \text{list}}}{\Gamma \mid\!-\ \text{length (tl l) : int}}\ \text{App}}{\Gamma \mid\!-\ 1 + \text{length (tl l) : int}}
$$

Polymorphic Example (7):App Rule

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

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

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

Polymorphic Example: (2) by BinOp

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

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

Polymorphic Example: (8)AppRule

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

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

Polymorphic Example: (8)AppRule

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

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

Polymorphic Example: (8)AppRule

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

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

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

Polymorphic Example: (10)BinOpRule

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

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

Polymorphic Example: (10)BinOpRule

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

$$\dfrac{\dfrac{}{\Gamma' \mid\!- 2 : \text{int}} \qquad \dfrac{}{\Gamma' \mid\!- [\,] : \text{int list}}}{\Gamma' \mid\!- (2 :: [\,]) : \text{int list}}$$

Polymorphic Example: (9)AppRule

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

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

Polymorphic Example: (9)AppRule

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

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

Polymorphic Example: (10)BinOpRule

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

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

Polymorphic Example: (10)BinOpRule

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

$$\dfrac{\dfrac{}{\Gamma' \text{ |- true : bool}} \qquad \dfrac{}{\Gamma' \text{ |- [] : bool list}}}{\Gamma' \text{ |- (true :: []) : bool list}}$$