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:

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

- 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

Application Examples

$$\frac{\Gamma \text{ |- print\_int : int} \rightarrow \text{unit} \qquad \Gamma \text{ |- 5 : int}}{\Gamma \text{ |- (print\_int 5) : unit}}$$

- e_1 = print_int, e_2 = 5,
- τ_1 = int, τ_2 = unit

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

- e_1 = map print_int, e_2 = [3; 7],
- τ_1 = int list, τ_2 = unit list

Fun Rule

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

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

Fun Examples

$$\frac{\{y : \text{int}\} + \Gamma \mid\!- \; y + 3 : \text{int}}{\Gamma \mid\!- \; \text{fun } y \text{ -> } y + 3 : \text{int} \rightarrow \text{int}}$$

$$\frac{\{f : \text{int} \rightarrow \text{bool}\} + \Gamma \mid\!- \; f\ 2 :: [\text{true}] : \text{bool list}}{\Gamma \mid\!- \; (\text{fun } f \text{ -> } f\ 2 :: [\text{true}]) \\ : (\text{int} \rightarrow \text{bool}) \rightarrow \text{bool list}}$$

(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:

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

Example

- Which rule do we apply?

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

Example

- Let rec rule: 2 {one : int list} |-

1 (let x = 2 in

{one : int list} |- fun y -> (x :: y :: one))

(1 :: one) : int list : int $\rightarrow$ int list

|- (let rec one = 1 :: one in

let x = 2 in

fun y -> (x :: y :: one)) : int $\rightarrow$ int list

Proof of 1

- Which rule?

{one : int list} |- (1 :: one) : int list

Proof of 1

- Application

$$\frac{\text{③ } \{\text{one : int list}\} \mid\!- \ ((::)\ 1) : \text{int list} \to \text{int list} \qquad \text{④ } \{\text{one : int list}\} \mid\!- \ \text{one : int list}}{\{\text{one : int list}\} \mid\!- \ (1 :: \text{one}) : \text{int list}}$$

Proof of 3

$$\dfrac{\dfrac{\text{Constants Rule}}{\{\text{one} : \text{int list}\} \mid\!- \ (::) : \text{int} \rightarrow \text{int list} \rightarrow \text{int list}} \quad \dfrac{\text{Constants Rule}}{\{\text{one} : \text{int list}\} \mid\!- \ 1 : \text{int}}}{\{\text{one} : \text{int list}\} \mid\!- \ ((::)\ 1) : \text{int list} \rightarrow \text{int list}}$$

Proof of 4

- Rule for variables

$$\frac{}{\{one : int\ list\} \mid\!- \ one{:}int\ list}$$

Proof of 2

5 {x:int; one : int list} |-

fun y ->

(x :: y :: one))

- Constant

: int → int list

{one : int list} |- 2:int

{one : int list} |- (let x = 2 in

fun y -> (x :: y :: one)) : int → int list

Proof of 5

$$\frac{?}{\{x:int;\ one : int\ list\}\ |\text{-}\ fun\ y \text{ -> } (x :: y :: one))\ :\ int \rightarrow int\ list}$$

Proof of 5

$$\frac{\dfrac{?}{\{y:int;\ x:int;\ one : int\ list\}\ |\text{-}\ (x :: y :: one) : int\ list}}{\{x:int;\ one : int\ list\}\ |\text{-}\ fun\ y \text{->} (x :: y :: one)) : int \rightarrow int\ list}$$

Proof of 5

⑥ ⑦

$$\dfrac{\dfrac{\{y{:}int;\ x{:}int;\ one{:}int\ list\} \ |\text{-}\ ((::)\ x){:}int\ list \rightarrow int\ list \qquad \{y{:}int;\ x{:}int;\ one{:}int\ list\} \ |\text{-}\ (y :: one) : int\ list}{\{y{:}int;\ x{:}int;\ one : int\ list\}\ |\text{-}\ (x :: y :: one) : int\ list}}{\{x{:}int;\ one : int\ list\}\ |\text{-}\ fun\ y \text{ -> } (x :: y :: one)) : int \rightarrow int\ list}$$

Proof of 6

Constant   Variable

$$\dfrac{}{\{\ldots\} \mid\!- \ (::) : \text{int} \rightarrow \text{int list} \rightarrow \text{int list}} \qquad \dfrac{}{\{\ldots;\ x{:}\text{int};\ldots\} \mid\!- \ x{:}\text{int}}$$

$$\overline{\{y{:}\text{int};\ x{:}\text{int};\ \text{one} : \text{int list}\} \mid\!- \ ((::)\ x) : \text{int list} \rightarrow \text{int list}}$$

Proof of 7

Pf of 6 [y/x] Variable

⋮

$$\frac{\dfrac{\vdots}{\{y{:}int;\ \ldots\} \mid\!-\ ((::)\ y) : int\ list \rightarrow int\ list} \qquad \dfrac{}{\{\ldots;\ one{:}\ int\ list\} \mid\!-\ one{:}\ int\ list}}{\{y{:}int;\ x{:}int;\ one : int\ list\} \mid\!-\ (y :: one) : int\ list}$$

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)
- 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
  - $\text{FreeVars}(\forall\alpha_1, \ldots, \alpha_n . \tau) = \text{FreeVars}(\tau) - \{\alpha_1, \ldots, \alpha_n\}$
- FreeVars(Γ) = all FreeVars of types in range of Γ

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

Polymorphic Typing Rules

- A *type judgement* has the form

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

  - 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:

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

- 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 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

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 Example

- Assume additional constants:
- 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

Polymorphic Example

- Show:

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

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

Polymorphic Example: Let Rec Rule

- Show: (1) (2)

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

Polymorphic Example (1)

- Show:

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

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

?

Γ|- if is_empty l then 0

else 1 + length (tl l) : int

Polymorphic Example (3):IfThenElse

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

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

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

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

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 ?

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

Polymorphic Example (4)

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

$$\dfrac{\dfrac{\text{By Const since } \alpha \text{ list -> bool is instance of } \forall\alpha.\ \alpha \text{ list -> bool}}{\Gamma \text{|- is\_empty : } \alpha \text{ list -> bool}} \qquad \dfrac{\text{By Variable } \Gamma(\text{l}) = \alpha \text{ list}}{\Gamma \text{|- l : } \alpha \text{ list}}}{\Gamma \text{|- is\_empty l : bool}}$$

- This finishes (4)

Polymorphic Example (5):Const

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

By Const Rule

$$\frac{}{\Gamma \mid\!\!- \; 0:\text{int}}$$

Polymorphic Example (6):Arith Op

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

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

Polymorphic Example (7):App Rule

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

$$\frac{\dfrac{\text{By Const}}{\Gamma \mid\!\text{-}\ \text{tl} : \alpha\ \text{list} \to \alpha\ \text{list}} \qquad \dfrac{\text{By Variable}}{\Gamma \mid\!\text{-}\ \text{l} : \alpha\ \text{list}}}{\Gamma \mid\!\text{-}\ (\text{tl l}) : \alpha\ \text{list}}$$

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

Polymorphic Example: (2) by ArithOp

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

$$\frac{\begin{array}{cc} (8) & (9) \\ \Gamma' \text{ |- } & \Gamma' \text{ |- } \\ \text{length ((::) 2 []) :int} & \text{length((::) true []) : int} \end{array}}{\begin{array}{l} \text{\{length: } \alpha.\ \alpha \text{ list -> int\}} \\ \text{|- length ((::) 2 []) + length((::) true []) : int} \end{array}}$$

Polymorphic Example: (8)AppRule

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

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

Polymorphic Example: (8)AppRule

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

By Var since int list -> int is instance of

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

(10)

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

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

$$\dfrac{\Gamma' \mid\!\!-((::)\ 2) : \text{int list} \to \text{int list} \qquad \dfrac{(11)}{\Gamma' \mid\!\!-\ [\,] : \text{int list}}}{\Gamma' \mid\!\!-\ ((::)\ 2\ [\,]) : \text{int list}}$$

Polymorphic Example: (11)AppRule

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

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

Polymorphic Example: (9)AppRule

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

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

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)

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

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

$$\dfrac{\Gamma' \mid\!- ((::)\text{true}):\text{bool list} \to \text{bool list} \qquad \dfrac{(13)}{\Gamma' \mid\!- []:\text{bool list}}}{\Gamma' \mid\!- ((::)\ \text{true}\ [])\ :\text{bool list}}$$

Polymorphic Example: (13)AppRule

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

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