### 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

10/3/23



# Type Inference

- Type inference: A program analysis to assign a type to an expression from the program context of the expression
  - Fully static type inference first introduced by Robin Miller in ML
  - Haskle, OCAML, SML all use type inference
    - Records are a problem for type inference

10/3/23



### Format of Type Judgments

A type judgement has the form

$$\Gamma$$
 |- exp :  $\tau$ 

- Γ is a typing environment
  - Supplies the types of variables (and function names when function names are not variables)
  - $\Gamma$  is a set of the form  $\{x:\sigma,\ldots\}$
  - For any x at most one  $\sigma$  such that  $(x : \sigma \in \Gamma)$
- exp is a program expression
- τ is a type to be assigned to exp
- |- pronounced "turnstyle", or "entails" (or "satisfies" or, informally, "shows")



#### **Axioms - Constants**

 $\Gamma \mid -n$ : int (assuming *n* is an integer constant)

$$\Gamma$$
 |- true : bool

 $\Gamma$  |- false : bool

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

10/3/23



#### Axioms – Variables (Monomorphic Rule)

Notation: Let  $\Gamma(x) = \sigma$  if  $x : \sigma \in \Gamma$ Note: if such  $\sigma$  exits, its unique

Variable axiom:

$$\overline{\Gamma \mid -x : \sigma}$$
 if  $\Gamma(x) = \sigma$ 



# Simple Rules - Arithmetic

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

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2 \quad (\oplus) : \tau_1 \to \tau_2 \to \tau_3}{\Gamma \vdash e_1 \oplus e_2 : \tau_3}$$

Special case: Relations (~∈ { < , > , =, <=, >= }):

$$\frac{\Gamma \mid -e_1 : \tau \quad \Gamma \mid -e_2 : \tau \quad (\sim) : \tau \to \tau \to bool}{\Gamma \mid -e_1 \quad \sim \quad e_2 : bool}$$

For the moment, think  $\tau$  is int

6

10/3/23

10/3/23



Example:  $\{x:int\} | -x + 2 = 3 : bool$ 

What do we need to show first?

$$\{x:int\} \mid -x + 2 = 3 : bool$$

10/3/23



Example:  $\{x:int\} | -x + 2 = 3 : bool$ 

What do we need for the left side?

$$\frac{\{x: int\} \mid -x+2: int \qquad \{x: int\} \mid -3: int \\ \{x: int\} \mid -x+2=3: bool}{}$$

10/3/23 8



Example:  $\{x:int\} | -x + 2 = 3 : bool$ 

How to finish?

10/3/23



Example:  $\{x:int\} \mid -x + 2 = 3 : bool$ 

Complete Proof (type derivation)

$$\frac{\frac{\text{Var}}{\{x:\text{int}\}\mid - x:\text{int}} \frac{\text{Const}}{\{x:\text{int}\}\mid - 2:\text{int}} \frac{\text{Const}}{\{x:\text{int}\}\mid - x + 2:\text{int}} \frac{\text{Const}}{\{x:\text{int}\}\mid - 3:\text{int}} \frac{\{x:\text{int}\}\mid - 3:\text{int}\}}{\{x:\text{int}\}\mid - x + 2 = 3:\text{bool}}$$

10/3/23 10



# Simple Rules - Booleans

#### Connectives

$$\frac{\Gamma \mid -e_1 : \mathsf{bool} \quad \Gamma \mid -e_2 : \mathsf{bool}}{\Gamma \mid -e_1 \&\& e_2 : \mathsf{bool}}$$

$$\frac{\Gamma \mid -e_1 : \mathsf{bool} \quad \Gamma \mid -e_2 : \mathsf{bool}}{\Gamma \mid -e_1 \mid \mid e_2 : \mathsf{bool}}$$

10/3/23



11

#### Type Variables in Rules

If\_then\_else rule:

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

- 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 e₁ of type τ₁ → τ₂ applied to an argument e₂ of type τ₁, the resulting expression e₁e₂ has type τ₂

10/3/23

13



### Fun Rule

- Can only do what rule allows!
- fun rule:

$$\frac{\{x \colon \tau_1\} + \Gamma \mid -e \colon \tau_2}{\Gamma \mid -\text{ fun } x \to e \colon \tau_1 \to \tau_2}$$

10/3/23 14



### Fun Examples

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

10/3/23

15

17



# (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:

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

10/3/23 16



#### Example

Which rule do we apply?

{} |- (let rec one = 1 :: one in let 
$$x = 2$$
 in fun  $y \rightarrow (x :: y :: one)$ ) : int  $\rightarrow$  int list

10/3/23

# Example



#### Proof of 1

Which rule?

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

10/3/23

#### Proof of 1

Binary Operator

{one: int list} |-

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

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

where ( :: ) : int  $\rightarrow$  int list  $\rightarrow$  int list

10/3/23 20



#### Proof of 1

Constant Rule

Variable Rule {one: int list} |-{one : int list} |-

1: int

one: int list

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

10/3/23



19

#### Proof of 2

Let Rule

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

fun y ->

(x :: y :: one))

: int  $\rightarrow$  int list {one : int list} |- 2:int

 $\{one : int list\} \mid - (let x = 2 in let x) \mid - (let x$ 

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

10/3/23



#### Proof of 2

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

Constant

(x :: y :: one))

{one : int list} |- 2:int : int  $\rightarrow$  int list

 $\{one : int list\} \mid - (let x = 2 in$ 

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

10/3/23

23

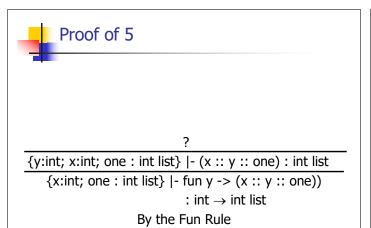


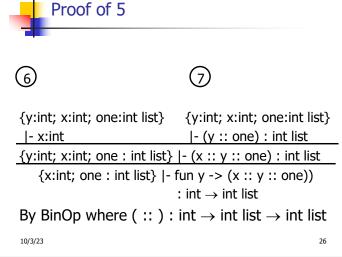
#### Proof of 5

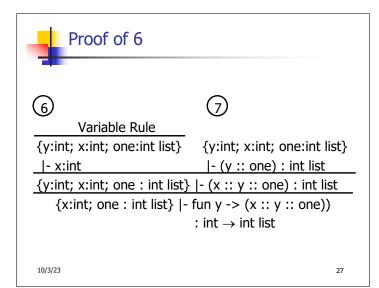
 $\{x:int; one : int list\} | -fun y -> (x :: y :: one)$ 

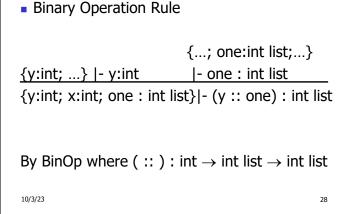
: int  $\rightarrow$  int list

10/3/23 24









Proof of 7



10/3/23

#### Proof of 7

Variable Rule

Variable Rule

{...; one:int list;...}

{y:int; ...} |- y:int |- one : int list

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

10/3/23 29



25

#### Curry - Howard Isomorphism

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



### Curry - Howard Isomorphism

Modus Ponens

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

Application

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

10/3/23

31

33

35



### Review: In Class Activity

### ACT 4

10/3/23 32



### Mea Culpa

- The above system can't handle polymorphism as in OCAML
- No type variables in type language (only metavariable 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

10/3/23



### Support for Polymorphic Types

- Monomorpic Types (τ):
  - Basic Types: int, bool, float, string, unit, ...
  - Type Variables:  $\alpha$ ,  $\beta$ ,  $\gamma$ ,  $\delta$ ,  $\epsilon$
  - Compound Types:  $\alpha \rightarrow \beta$ , int \* string, bool list, ...
- Polymorphic Types:
  - Monomorphic types τ
  - Universally quantified monomorphic types
  - $\forall \alpha_1, \dots, \alpha_n . \tau$
  - Can think of  $\tau$  as same as  $\forall . \tau$

10/3/23 34



# **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( $\forall \alpha_1, ..., \alpha_n \cdot \tau$ ) = FreeVars( $\tau$ ) { $\alpha_1, ..., \alpha_n$  }
- FreeVars( $\Gamma$ ) = all FreeVars of types in range of  $\Gamma$

10/3/23 36

10/3/23



### Monomorphic to Polymorphic

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

10/3/23 37



### Polymorphic Typing Rules

A type judgement has the form

$$\Gamma$$
 |- 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

723

38



# Polymorphic Let and Let Rec

let rule:

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

let rec rule:

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

10/3/23



#### Polymorphic Variables (Identifiers)

Variable axiom:

$$\overline{\Gamma \mid - X : \varphi(\tau)}$$
 if  $\Gamma(x) = \forall \alpha_1, \dots, \alpha_n \cdot \tau$ 

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

$$\overline{\Gamma \mid - x \colon \tau} \quad \text{if } \Gamma(x) = \tau$$

Constants treated same way

3/23



# Fun Rule Stays the Same

• fun rule:

$$\frac{\{x \colon \tau_1\} + \Gamma \mid -e \colon \tau_2}{\Gamma \mid -\text{fun } x \to e \colon \tau_1 \to \tau_2}$$

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

4

# Polymorphic Example

- Assume additional constants and primitive operators:
- hd : $\forall \alpha$ .  $\alpha$  list ->  $\alpha$
- tl:  $\forall \alpha$ .  $\alpha$  list ->  $\alpha$  list
- is\_empty :  $\forall \alpha$ .  $\alpha$  list -> bool
- (::) :  $\forall \alpha$ .  $\alpha \rightarrow \alpha$  list  $\rightarrow \alpha$  list
- [] : ∀α. α list

10/3/23

10/3/23

41

42



### Polymorphic Example

Show:

?

10/3/23 43

# 4

### Polymorphic Example: Let Rec Rule

Show: (1) (2)
{length:α list -> int} {length:∀α. α list -> int}
|- fun I -> ... |- length (2 :: []) +
: α list -> int | length(true :: []) : int

{} |- let rec length =
fun I -> if is\_empty | then 0
else 1 + length (tl |)

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

10/3/23 44



### Polymorphic Example (1)

Show:

?

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

10/3/23 4



### Polymorphic Example (1): Fun Rule

Show: (3)
{length:α list -> int, I: α list } |if is\_empty I then 0
else length (hd I) + length (tl I) : int
{length:α list -> int} |-

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

10/3/23 46



### Polymorphic Example (3)

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

?

 $\Gamma$ |- if is\_empty | then 0 else 1 + length (tl |) : int

10/3/23 47



### Polymorphic Example (3):IfThenElse

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

(4) (5) (6) 
$$\Gamma$$
 | - is\_empty |  $\Gamma$  | - 0:int  $\Gamma$  | - 1 + length (tl |) : bool : int

 $\Gamma$ |- if is\_empty | then 0 else 1 + length (tl |) : int



### Polymorphic Example (4)

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

 $\Gamma$ |- is\_empty | : bool

10/3/23

# Polymorphic Example (4):Application

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

?

?

 $\Gamma$ |- is\_empty :  $\alpha$  list -> bool  $\Gamma$ |-|:  $\alpha$  list

 $\Gamma$ |- is\_empty | : bool

10/3/23 50



### Polymorphic Example (4)

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

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

 $\Gamma$ |- is\_empty :  $\alpha$  list -> bool  $\Gamma$ |-|:  $\alpha$  list

 $\Gamma$ |- is\_empty | : bool

?

10/3/23



### Polymorphic Example (4)

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

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

 $\Gamma$ |- is\_empty :  $\alpha$  list -> bool  $\Gamma$ |-|:  $\alpha$  list

 $\Gamma$ |- is\_empty I : bool

This finishes (4)

10/3/23 52



# Polymorphic Example (5):Const

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

By Const Rule

 $\Gamma$ |- 0:int



# Polymorphic Example (6):Arith Op

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

By Const

By Variable

Γ|- length

(7):  $\alpha$  list -> int  $\Gamma$ |- (tl l) :  $\alpha$  list

 $\Gamma$  - 1:int  $\Gamma$  | - length (tl l) : int

 $\Gamma$ |-1 + length (tl l) : int

10/3/23

10/3/23

53

54



# Polymorphic Example (7):App Rule

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

By Const

By Variable

$$\Gamma$$
|- tl :  $\alpha$  list ->  $\alpha$  list

$$\Gamma$$
|-|:  $\alpha$  list

$$\Gamma$$
|- (tl l) :  $\alpha$  list

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

10/3/23

55



### Polymorphic Example: (2) by ArithOp

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

(9) Γ' |-

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

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

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

10/3/23 56



# Polymorphic Example: (8)AppRule

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

 $\Gamma'$  |- length : int list ->int  $\Gamma'$  |- (2 :: []) :int list

 $\Gamma'$  |- length (2 :: []) :int

10/3/23

57