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 Declarations

- *Type declarations*: explicit assignment of types to variables (signatures to functions) in the code of a program
  - Must be checked in a strongly typed language
  - Often not necessary for strong typing or even static typing (depends on the type system)

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

Format of Type Judgments

- A *type judgement* has the form

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

- Γ is a typing environment
  - Supplies the types of variables (and function names when function names are not variables)
  - Γ is a set of the form $\{ x : \sigma , . . . \}$
  - For any x at most one σ 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

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

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

- 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:

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

Simple Rules - Arithmetic

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

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

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

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

For the moment, think τ is int

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

What do we need to show first?

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

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

What do we need for the left side?

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

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

How to finish?

$$\dfrac{\dfrac{\{x:int\} \mid\!- x:int \quad \{x:int\} \mid\!- 2:int}{\{x : int\} \mid\!- x + 2 : int}\,\text{AO} \qquad \{x:int\} \mid\!- 3 :int}{\{x:int\} \mid\!- x + 2 = 3 : bool}\,\text{Rel}$$

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

Complete Proof (type derivation)

$$\dfrac{\dfrac{\dfrac{}{\{x:int\} \mid\!- \; x:int}\text{Var} \quad \dfrac{}{\{x:int\} \mid\!- \; 2:int}\text{Const}}{\{x : int\} \mid\!- \; x + 2 : int}\text{AO} \quad \dfrac{}{\{x:int\} \mid\!- \; 3 :int}\text{Const}}{\{x:int\} \mid\!- \; x + 2 = 3 : bool}\text{Rel}$$

Simple Rules - Booleans

Connectives

$$\frac{\Gamma \text{ |- } e_1 : \text{bool} \qquad \Gamma \text{ |- } e_2 : \text{bool}}{\Gamma \text{ |- } e_1 \;\&\&\; e_2 : \text{bool}}$$

$$\frac{\Gamma \text{ |- } e_1 : \text{bool} \qquad \Gamma \text{ |- } e_2 : \text{bool}}{\Gamma \text{ |- } e_1 \;||\; e_2 : \text{bool}}$$

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

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

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

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

(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?

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

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

Example

- Let rec rule: ② {one : int list} |- (let x = 2 in fun y -> (x :: y :: one)) : int → int list

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

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

|- (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

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

Proof of 3

Constants Rule   Constants Rule

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

Proof of 4

- Rule for variables

$$\frac{}{\{\text{one : int list}\} \ |\!\!- \ \text{one:int list}}$$

Proof of 2

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

- Constant

{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

?

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

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

$$\frac{}{\{...\} \;|\text{-}\; (::) : \text{int} \rightarrow \text{int list} \rightarrow \text{int list}}$$

Variable

$$\frac{}{\{...;\ x:\text{int};...\} \;|\text{-}\; x:\text{int}}$$

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

Proof of 7

Pf of 6 [y/x] Variable

⋮

{y:int; …} |- ((::) y) :int list→ int list

{…; 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

- Function space arrow corresponds to implication; application corresponds to modus ponens

Curry - Howard Isomorphism

- Modus Ponens

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

- Application

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

Mea Culpa

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

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:

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

- let rec rule:

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

Polymorphic Variables (Identifiers)

Variable axiom:

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

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

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

- 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 :$\forall\alpha.\ \alpha$ 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: Let Rec Rule

-

?

{} |- 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)

(1)

$\{length{:}\alpha \text{ list} \to \text{int}\}$ |- fun l -> …

: α list -> int

(2)

$\{length{:}\forall\alpha.\ \alpha \text{ list} \to \text{int}\}$ |- 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 (1)

- Show:

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

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

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

Polymorphic Example (4)

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

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

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 ?

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

Polymorphic Example (4)

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

By Const since α list -> bool is instance 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

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

Polymorphic Example (6):Arith Op

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

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

Polymorphic Example (7):App Rule

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

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

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

Polymorphic Example: (2) by ArithOp

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

(8) (9)

Γ' |- length (2 :: []) :int $\quad$ Γ' |- length(true :: []) : int

---

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

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

Polymorphic Example: (8)AppRule

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

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

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)

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

Γ' |- 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

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

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

$$\dfrac{\dfrac{\text{By Const since } \alpha \text{ list is instance of } \forall\alpha.\ \alpha \text{ list}}{\Gamma' \mid\!- (::) : \text{int -> int list -> int list}} \qquad \dfrac{\text{By Const}}{\Gamma' \mid\!- 2 : \text{int}}}{\Gamma' \mid\!- ((::)\ 2) : \text{int list -> int list}}$$

Polymorphic Example: (9)AppRule

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

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

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)

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

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

$$\dfrac{\dfrac{(13)}{\Gamma' \text{ |-((::)true):bool list ->bool list}} \quad \dfrac{}{\Gamma' \text{ |- []:bool list}}}{\Gamma' \text{ |- ((::) true []) :bool list}}$$

Polymorphic Example: (13)AppRule

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

By Const since bool list
is instance of ∀α. α list

By Const

Γ' |- (::):bool ->bool list ->bool list

Γ' |- true : bool

Γ' |- ((::) true) : bool list -> bool list