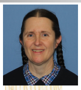
Programming Languages and Compilers (CS 421)

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

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 $\{ x:\sigma, \ldots \}$
  - 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 (Monomorphic)

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

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

- 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:

$$\overline{\Gamma \text{ |- } x : \sigma} \quad \text{if } \Gamma(x) = \sigma$$

Simple Rules – Arithmetic (Mono)

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

$$\frac{\Gamma \text{ |- } e_1 : \tau_1 \quad \Gamma \text{ |- } e_2 : \tau_2 \quad (\oplus) : \tau_1 \rightarrow \tau_2 \rightarrow \tau_3}{\Gamma \text{ |- } e_1 \oplus e_2 : \tau_3}$$

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

$$\frac{\Gamma \text{ |- } e_1 : \tau \quad \Gamma \text{ |- } e_2 : \tau \quad (\sim) : \tau \rightarrow \tau \rightarrow \text{bool}}{\Gamma \text{ |- } e_1 \sim e_2 : \text{bool}}$$

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?

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

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

How to finish?

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

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

Complete Proof (type derivation)

$$\frac{\dfrac{\dfrac{}{\{x:int\} \mid\!- x:int}\text{Var} \quad \dfrac{}{\{x:int\} \mid\!- 2:int}\text{Const}}{\{x : int\} \mid\!- x + 2 : int}\text{Bin} \qquad \dfrac{}{\{x:int\} \mid\!- 3 : int}\text{Const}}{\{x:int\} \mid\!- x + 2 = 3 : bool}\text{Bin}$$

Simple Rules - Booleans

Connectives

$$\frac{\Gamma \mid\!- e_1 : bool \qquad \Gamma \mid\!- e_2 : bool}{\Gamma \mid\!- e_1 \,\&\&\, e_2 : bool}$$

$$\frac{\Gamma \mid\!- e_1 : bool \qquad \Gamma \mid\!- e_2 : bool}{\Gamma \mid\!- e_1 \,||\, e_2 : bool}$$

Type Variables in Rules

- If_then_else rule:

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

- 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

Example derivation: if-then-else-

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

$$\frac{\begin{array}{c}\Gamma \mid\!- \text{(fun y -> y > 3) x} \\ : bool\end{array} \quad \begin{array}{c}\Gamma \mid\!- x+2 \\ : int\end{array} \quad \begin{array}{c}\Gamma \mid\!- \text{int\_of\_float y} \\ : int\end{array}}{\Gamma \mid\!- \text{if (fun y -> y > 3) x then x + 2 else int\_of\_float y} : int}$$

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 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

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

$$\frac{\Gamma \mid\!- \text{(fun y -> y > 3)} : \text{int -> bool} \quad \Gamma \mid\!- \text{x : int}}{\Gamma \mid\!- \text{(fun y -> y > 3) x : bool}}$$

Fun Rule

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

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

Fun Examples

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

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

(Monomorphic) Let and Let Rec

- let rule:

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

- let rec rule:

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

Example

- Which rule do we apply?

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

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

ROW: ① and ② above the line

{} |- (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

- Binary Operator

③ {one : int list} |- 1: int   ④ {one : int list} |- one : int list

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

where (::) : int → int list → int list

Proof of 1

③ Constant Rule
{one : int list} |- 1: int

④ Variable Rule
{one : int list} |- one : int list

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

750 minutes

Proof of 2

- Let Rule

{one : int list} |- 2:int   {x:int; one : int list} |- fun y -> (x :: y :: one)) : int → int list

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

Proof of 2

- Constant

{one : int list} |- 2:int

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

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

Proof of 5

?

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

By the Fun Rule

Proof of 5

⑥ ? — {y:int; x:int; one:int list} |- x:int

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

By BinOp where (::) : int → int list → int list

Proof of 6

⑥ Variable Rule — {y:int; x:int; one:int list} |- x:int

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 7

- Binary Operation Rule

? — {y:int; ...} |- y:int

? — {...; one:int list;...} |- one : int list

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

By BinOp where (::) : int → int list → int list

Proof of 7

Variable Rule ——— {...; one:int list;...}

Variable Rule ——— {y:int; ...} |- y:int

|- 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

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