Programming Languages and Compilers (CS 421)

Elsa L Gunter
2112 SC, UIUC
http://www.cs.illinois.edu/class/cs421/

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 functions
  - Γ is a list of the form [x : σ , . . .]
- exp is a program expression
- τ is a type to be assigned to exp
- |- pronounced "turnstyle", or "entails" (or "satisfies")

Axioms - Constants

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

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

- These rules are true with any typing environment
- n is a meta-variable

Axioms - Variables

Notation: Let $\Gamma(x) = \sigma$ if $x : \sigma \in \Gamma$ and there is no $x : \tau$ to the left of $x : \sigma$ in Γ

Variable axiom:

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

Simple Rules - Arithmetic

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

$$\frac{\Gamma \text{ |- } e_1 : \text{int} \qquad \Gamma \text{ |- } e_2 : \text{int}}{\Gamma \text{ |- } e_1 \oplus e_2 : \text{int}}$$

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

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

Simple Rules - Booleans

Connectives

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

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

Type Variables in Rules

- If_then_else rule:

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

- 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:

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

- If you have a function expression e_1 of type $\tau_1 \rightarrow \tau_2$ applied to an argument of type τ_1, the resulting expression 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

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

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

(Monomorphic) Let and Let Rec

- let rule:

$$\frac{\Gamma \text{ |- } e_1 : \tau_1 \quad [x : \tau_1] + \Gamma \text{ |- } e_2 : \tau_2}{\Gamma \text{ |- (let x = } e_1 \text{ in } e_2) : \tau_2}$$

- let rec rule:

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

Example

- Which rule do we apply?

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

Example

- Let rec rule:

① [one : int list] |- (1 :: one) : int list

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

③ [one : int list] |- ((::) 1): int list→ int list

④ [one : int list] |- one : int list

[one : int list] |- (1 :: one) : int list

Proof of 3

Constants Rule

[one : int list] |- (::) : int → int list→ int list

Constants Rule

[one : int list] |- 1 : int

[one : int list] |- ((::) 1) : int list → int list

Proof of 4

- Rule for variables

[one : int list] |- one: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

Proof of 5

(6) (7)

[y:int; x:int; one : int list] |- [y:int; x:int; one : int list] |-
((::) x):int list→ 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 Variable

[…] |- (::) [...; x:int;...] |- x:int
: int→ int list→ int list

[y:int; x:int; one : int list] |- ((::) x)
:int list→ int list

Proof of 7

Pf of 6 [y/x] Variable

⋮

[y:int; …] |- ((::) y) […; one: int list] |-
:int list→ 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

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

Curry - Howard Isomorphism

- Modus Ponens

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

- Application

$$\frac{\Gamma \text{ |- } e_1 : \alpha \rightarrow \beta \quad \Gamma \text{ |- } e_2 : \alpha}{\Gamma \text{ |- } (e_1\ e_2) : \beta}$$

Mia Copa

- The above system can't handle polymorphism as in OCAML
- No type variables in type language (only meta-variable in the logic)
- 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
- Gen(τ, Γ) = $\forall \alpha_1, \ldots, \alpha_n \,.\, \tau$ where $\{\alpha_1, \ldots, \alpha_n\}$ = freeVars(τ) – freeVars(Γ)

Polymorphic Typing Rules

- A *type judgement* has the form

  Γ |- exp : τ
  - 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:

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

- let rec rule:

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

Polymorphic Variables (Identifiers)

Variable axiom:

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

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

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

- 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

- Show:

?

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

Polymorphic Example: Let Rec Rule

- Show: (1) (2)

{length:α list -> int} {length: $\alpha.\ \alpha$ list -> int}
|- fun l -> … |- length ((::) 2 []) +
: α list -> int ((::) true []) : int

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

Polymorphic Example (1)

- Show:

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

Polymorphic Example (1): Fun Rule

- Show: (3)

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

Polymorphic Example (3)

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

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

Polymorphic Example (3):IfThenElse

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

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

Polymorphic Example (4)

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

$$\frac{?}{\Gamma \vdash is\_empty\ l : bool}$$

Polymorphic Example (4):Application

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

$$\frac{? \quad ?}{\Gamma \vdash is\_empty : \alpha\ list \rightarrow bool \quad \Gamma \vdash l : \alpha\ list}{\Gamma \vdash is\_empty\ l : bool}$$

Polymorphic Example (4)

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

By Const since α list -> bool
instance of α. α list -> bool ?

$$\frac{\Gamma \vdash \text{is\_empty} : \alpha\ \text{list} \to \text{bool} \qquad \Gamma \vdash l : \alpha\ \text{list}}{\Gamma \vdash \text{is\_empty}\ l : \text{bool}}$$

Polymorphic Example (4)

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

By Const since α list -> bool
instance of α. α list -> bool
By Variable
$\Gamma(l) = \alpha$ list

$$\frac{\Gamma \vdash \text{is\_empty} : \alpha\ \text{list} \to \text{bool} \qquad \Gamma \vdash l : \alpha\ \text{list}}{\Gamma \vdash \text{is\_empty}\ l : \text{bool}}$$

- This finishes (4)

Polymorphic Example (5):Const

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

By Const Rule

$$\frac{}{\Gamma \vdash 0 : \text{int}}$$

Polymorphic Example (6):Arith Op

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

$$\frac{\dfrac{}{\Gamma \vdash 1 : \text{int}}\ \text{By Const} \qquad \dfrac{\dfrac{}{\Gamma \vdash \text{length} : \alpha\ \text{list} \to \text{int}}\ \text{By Variable} \qquad \dfrac{}{\Gamma \vdash (\text{tl}\ l) : \alpha\ \text{list}}\ (7)}{\Gamma \vdash \text{length}\ (\text{tl}\ l) : \text{int}}}{\Gamma \vdash 1 + \text{length}\ (\text{tl}\ l) : \text{int}}$$