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:

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

- 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

Application Examples

$$\frac{\Gamma \mid\!- \text{print\_int} : \text{int} \rightarrow \text{unit} \quad \Gamma \mid\!- 5 : \text{int}}{\Gamma \mid\!- (\text{print\_int } 5) : \text{unit}}$$

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

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

- 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

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

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

(Monomorphic) Let and Let Rec

- let rule:

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

- let rec rule:

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

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:

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

Proof of 1

- Which rule?

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

Proof of 1

- Application

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

Proof of 3

$$\frac{\dfrac{\text{Constants Rule}}{\text{\{one : int list\} |- (::) : int} \rightarrow \text{int list} \rightarrow \text{int list}} \qquad \dfrac{\text{Constants Rule}}{\text{\{one : int list\} |- 1 : int}}}{\text{\{one : int list\} |- ((::) 1) : int list} \rightarrow \text{int list}}$$

Proof of 4

- Rule for variables

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

Proof of 2

- Constant

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

?

{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 {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   Variable

{...} |- (::) : int→ int list→ int list

{...; x:int;...} |- x:int

{y:int; x:int; one : int list} |- ((::) x) :int list→ 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
  - 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
- $Gen(\tau, \Gamma) = \forall \alpha_1, \ldots, \alpha_n . \tau$ where $\{\alpha_1, \ldots, \alpha_n\} = freeVars(\tau) - freeVars(\Gamma)$

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:

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

- let rec rule:

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

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:

?

{} |- 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:α list -> int} |- fun l -> … : α list -> int

(2) {length:$\forall\alpha$. α list -> 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:

?

{length:α list -> int} |-
fun l -> if is_empty l then 0
else 1 + length (tl l)
: α 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

(4) Γ|- is_empty l : bool

(5) Γ|- 0:int

(6) Γ|- 1 + length (tl l) : int

Γ|- 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

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

Polymorphic Example (4)

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

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

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

Polymorphic Example (4)

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

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

By Variable Γ(l) = α list

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

- This finishes (4)

Polymorphic Example (5):Const

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

By Const Rule

$$\frac{}{\Gamma |\text{- 0:int}}$$

Polymorphic Example (6):Arith Op

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

By Const; By Variable; (7)

$$\frac{\dfrac{}{\Gamma |\text{- 1:int}} \quad \dfrac{\dfrac{}{\Gamma |\text{- length : } \alpha \text{ list -> int}} \quad \Gamma |\text{- (tl l) : } \alpha \text{ list}}{\Gamma |\text{- length (tl l) : int}}}{\Gamma |\text{- 1 + length (tl l) : int}}$$

Polymorphic Example (7):App Rule

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

By Const — Γ|- tl : α list -> α list

By Variable — Γ|- l : α list

Γ|- (tl l) : α 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:$\forall$ α. α list -> int}
- Show:

(8) Γ' |- length ((::) 2 []) :int

(9) Γ' |- length((::) true []) : int

{length: α. α list -> int}
|- length ((::) 2 []) + length((::) true []) : int

Polymorphic Example: (8)AppRule

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

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

Γ' |- length ((::) 2 []) :int

Polymorphic Example: (8)AppRule

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

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

Γ' |- length : int list ->int   (10) Γ' |- ((::)2 []):int list

Γ' |- length ((::) 2 []) :int

Polymorphic Example: (10)AppRule

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

(11) Γ' |-((::) 2) : int list -> int list Γ' |- [] : int list

Γ' |- ((::) 2 []) :int list

Polymorphic Example: (11)AppRule

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

Γ' |- (::) : int -> int list -> int list   By Const — Γ' |- 2 : int

Γ' |- ((::) 2) : int list -> int list

Polymorphic Example: (9)AppRule

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

Γ' |- length:bool list ->int   Γ' |- ((::) true []):bool list
―――――――――――――――――――――――
Γ' |- length ((::) true []) :int

Polymorphic Example: (9)AppRule

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

By Var since bool list -> int is instance of ∀α. α list -> int

(12)

Γ' |- length:bool list ->int   Γ' |- ((::) true []):bool list
―――――――――――――――――――――――
Γ' |- length ((::) true []) :int

Polymorphic Example: (12)AppRule

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

(13)

Γ' |-((::)true):bool list ->bool list   Γ' |- []:bool list
―――――――――――――――――――――――
Γ' |- ((::) 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