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

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

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:

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

- let rec rule:

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

Polymorphic Variables (Identifiers)

Variable axiom:

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

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

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

- Constants treated same way

Fun Rule Stays the Same

- fun rule:

$$\frac{[x : \tau_1] + \Gamma \vdash e : \tau_2}{\Gamma \vdash \text{fun } x \text{ -> } e : \tau_1 \rightarrow \tau_2}$$

- 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

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

Polymorphic Example (4)

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

$$\frac{\dfrac{\text{By Const since } \alpha \text{ list -> bool is instance of } \forall\alpha.\ \alpha \text{ list -> bool}}{\Gamma|\text{- is\_empty : } \alpha \text{ list -> bool}} \qquad \dfrac{?}{\Gamma|\text{- l : } \alpha \text{ list}}}{\Gamma|\text{- is\_empty l : bool}}$$

Polymorphic Example (4)

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

$$\frac{\dfrac{\text{By Const since } \alpha \text{ list -> bool is instance of } \forall\alpha.\ \alpha \text{ list -> bool}}{\Gamma|\text{- is\_empty : } \alpha \text{ list -> bool}} \qquad \dfrac{\text{By Variable } \Gamma(\text{l}) = \alpha \text{ list}}{\Gamma|\text{- l : } \alpha \text{ list}}}{\Gamma|\text{- is\_empty l : bool}}$$

- 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

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

Polymorphic Example (7):App Rule

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

$$\dfrac{\dfrac{\text{By Const}}{\Gamma \vdash (\text{tl l}) : \alpha\ \text{list} \to \alpha\ \text{list}} \qquad \dfrac{\text{By Variable}}{\Gamma \vdash \text{l} : \alpha\ \text{list}}}{\Gamma \vdash (\text{tl l}) : \alpha\ \text{list}}$$

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

Polymorphic Example: (2) by ArithOp

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

$$\dfrac{\begin{matrix}(8) \\ \Gamma' \vdash \\ \text{length } ((::)\ 2\ [\,]) : \text{int}\end{matrix} \qquad \begin{matrix}(9) \\ \Gamma' \vdash \\ \text{length}((::)\ \text{true}\ [\,]) : \text{int}\end{matrix}}{\begin{matrix}\{\text{length: } \alpha.\ \alpha\ \text{list} \to \text{int}\} \\ \vdash \text{length } ((::)\ 2\ [\,]) + \text{length}((::)\ \text{true}\ [\,]) : \text{int}\end{matrix}}$$

Polymorphic Example: (8)AppRule

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

$$\dfrac{\Gamma' \vdash \text{length} : \text{int list} \to \text{int} \qquad \Gamma' \vdash ((::)2\ [\,]) : \text{int list}}{\Gamma' \vdash \text{length } ((::)\ 2\ [\,]) : \text{int}}$$

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

$$\dfrac{\dfrac{}{\Gamma' \vdash \text{length} : \text{int list} \to \text{int}} \qquad \begin{matrix}(10) \\ \Gamma' \vdash ((::)2\ [\,]) : \text{int list}\end{matrix}}{\Gamma' \vdash \text{length } ((::)\ 2\ [\,]) : \text{int}}$$

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

$$\dfrac{\begin{matrix}(11) \\ \Gamma' \vdash ((::)\ 2) : \text{int list} \to \text{int list}\end{matrix} \qquad \dfrac{}{\Gamma' \vdash [\,] : \text{int list}}}{\Gamma' \vdash ((::)\ 2\ [\,]) : \text{int list}}$$

Polymorphic Example: (11)AppRule

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

$$\dfrac{\dfrac{\forall\alpha.\ \alpha\ \text{list}}{\Gamma' \vdash (::) : \text{int} \to \text{int list} \to \text{int list}} \qquad \dfrac{\text{By Const}}{\Gamma' \vdash 2 : \text{int}}}{\Gamma' \vdash ((::)\ 2) : \text{int list} \to \text{int list}}$$

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