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

Axioms - Constants

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

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

- 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:

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

Simple Rules - Arithmetic

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

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2 \quad (\oplus) : \tau_1 \to \tau_2 \to \tau_3}{\Gamma \vdash e_1 \oplus e_2 : \tau_3}$$

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

$$\frac{\Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1 \sim e_2 : \text{bool}}$$

For the moment, think τ is int

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

What do we need to show first?

$$\{x : \text{int}\} \vdash x + 2 = 3 : \text{bool}$$

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

What do we need for the left side?

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

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

How to finish?

$$\dfrac{\dfrac{\{x{:}int\} \vdash x{:}int \quad \{x{:}int\} \vdash 2{:}int}{\{x : int\} \vdash x + 2 : int}\,AO \qquad \{x{:}int\} \vdash 3 : int}{\{x{:}int\} \vdash x + 2 = 3 : bool}\,Rel$$

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

Complete Proof (type derivation)

$$\dfrac{\dfrac{\dfrac{}{\{x{:}int\} \vdash x{:}int}\,Var \quad \dfrac{}{\{x{:}int\} \vdash 2{:}int}\,Const}{\{x : int\} \vdash x + 2 : int}\,AO \qquad \dfrac{}{\{x{:}int\} \vdash 3 : int}\,Const}{\{x{:}int\} \vdash x + 2 = 3 : bool}\,Rel$$

Simple Rules - Booleans

Connectives

$$\dfrac{\Gamma \vdash e_1 : bool \quad \Gamma \vdash e_2 : bool}{\Gamma \vdash e_1 \,\&\&\, e_2 : bool}$$

$$\dfrac{\Gamma \vdash e_1 : bool \quad \Gamma \vdash e_2 : bool}{\Gamma \vdash e_1 \,||\, e_2 : bool}$$

Type Variables in Rules

- If_then_else rule:

$$\dfrac{\Gamma \vdash e_1 : bool \quad \Gamma \vdash e_2 : \tau \quad \Gamma \vdash e_3 : \tau}{\Gamma \vdash (\text{if } e_1 \text{ then } e_2 \text{ else } e_3) : \tau}$$

- 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:

$$\dfrac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash (e_1\ e_2) : \tau_2}$$

- If you have a function expression e_1 of type $\tau_1 \to \tau_2$ applied to an argument e_2 of type τ_1, the resulting expression $e_1 e_2$ has type τ_2

Application Examples

$$\dfrac{\Gamma \vdash print\_int : int \to unit \quad \Gamma \vdash 5 : int}{\Gamma \vdash (print\_int\ 5) : unit}$$

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

$$\dfrac{\Gamma \vdash map\ print\_int : int\ list \to unit\ list \quad \Gamma \vdash [3;7] : int\ list}{\Gamma \vdash (map\ print\_int\ [3;\ 7]) : unit\ list}$$

- 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:

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

Fun Examples

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

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

(Monomorphic) Let and Let Rec

- let rule:

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

- let rec rule:

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

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:

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

Proof of 1

- Which rule?

$$\text{\{one : int list\} |- (1 :: one) : int list}$$

Proof of 1

- Primitive Operation

(::): int →int list→ int list

③ {one : int list} |- 1:int

④ {one : int list} |- one : int list

PO

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

Proof of 1

(::): int →int list→ int list

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

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

PO

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

Proof of 2

?

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

Let

{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

Fun

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

Proof of 5

- Primitive Operation

(::): int →int list→ int list

⑥ ⑦

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

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

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

: int → int list

Proof of 5

- Primitive Operation

(::): int →int list→ int list

⑥ Variable Rule ⑦

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

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

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

: int → int list

Proof of 7

(::): int →int list→ int list

Variable Rule Variable Rule

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

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

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

$$\frac{\Gamma \vdash e_1 : \alpha \rightarrow \beta \quad \Gamma \vdash e_2 : \alpha}{\Gamma \vdash (e_1\ e_2) : \beta}$$

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(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:

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

- let rec rule:

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

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:

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

- Types τ_1, τ_2 monomorphic
- Function argument must always be used at same type in function body

Polymorphic Example

- Assume additional constants:
- hd :∀α. α list -> α
- tl: ∀α. α list -> α list
- is_empty : ∀α. α list -> bool
- :: : ∀α. α -> α list -> α list
- [] : ∀α. α 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)

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

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

Polymorphic Example (3):IfThenElse

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

$$\frac{\Gamma\text{|- is\_empty l : bool} \quad \Gamma\text{|- 0:int} \quad \Gamma\text{|- 1 + length (tl l) : int}}{\Gamma\text{|- if is\_empty l then 0 else 1 + length (tl l) : int}}$$

(4) (5) (6)

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

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

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 ?

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

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

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

- This finishes (4)

Polymorphic Example (5):Const

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

By Const Rule

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

Polymorphic Example (6):Arith Op

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

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

Polymorphic Example (7):App Rule

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

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

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:

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

Polymorphic Example: (8)AppRule

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

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

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

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

Polymorphic Example: (10)AppRule

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

(11)

$$\frac{\Gamma' \text{ |-((::) 2) : int list -> int list} \quad \Gamma' \text{ |- [] : int list}}{\Gamma' \text{ |- ((::) 2 []) :int list}}$$

Polymorphic Example: (11)AppRule

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

$$\frac{\dfrac{\Gamma' \text{ |- (::) : int -> int list -> int list}}{\text{(11)}} \quad \dfrac{\Gamma' \text{ |- 2 : int}}{\text{By Const}}}{\Gamma' \text{ |- ((::) 2) : int list -> int list}}$$

Polymorphic Example: (9)AppRule

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

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

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)

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

Polymorphic Example: (12)AppRule

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

(13)

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

Polymorphic Example: (13)AppRule

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

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

$$\frac{\dfrac{\Gamma' \text{ |- (::):bool ->bool list ->bool list}}{\text{(13)}} \quad \dfrac{\Gamma' \text{ |- true : bool}}{\text{By Const}}}{\Gamma' \text{ |- ((::) true) : bool list -> bool list}}$$