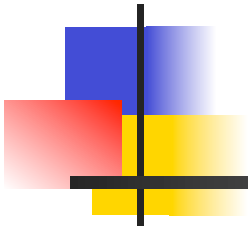


Programming Languages and Compilers (CS 421)



Dennis Griffith
0207 SC, UIUC

<http://www.cs.illinois.edu/class/cs421/>

Based in part on slides by Mattox Beckman, as updated by Vikram Adve, Gul Agha, and Elsa Gunter



Format of Type Judgments

- A *type judgement* has the form

$$\Gamma \vdash \text{exp} : \tau$$

- Γ is a typing environment
 - Supplies the types of variables and functions
 - Γ is a list of the form $[x : \sigma , \dots]$
- exp is a program expression
- τ is a type to be assigned to exp
- \vdash pronounced “turnstile”, or “entails” (or “satisfies”)



Axioms - Constants

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

$\overline{\quad}$
 $\vdash \text{true} : \text{bool}$

$\overline{\quad}$
 $\vdash \text{false} : \text{bool}$

- 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:

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



Simple Rules - Arithmetic

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

$$\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 \oplus e_2 : \text{int}}$$

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

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



Simple Rules - Booleans

Connectives

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

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



Type Variables in Rules

- If_then_else rule:

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

- 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:

$$\frac{\Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash (e_1 e_2) : \tau_2}$$

- If you have a function expression e_1 of type $\tau_1 \rightarrow \tau_2$ 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:

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



Fun Examples

$$\frac{[y : \text{int}] + \Gamma \vdash y + 3 : \text{int}}{\Gamma \vdash \text{fun } y \rightarrow y + 3 : \text{int} \rightarrow \text{int}}$$
$$\frac{[f : \text{int} \rightarrow \text{bool}] + \Gamma \vdash f \ 2 :: [\text{true}] : \text{bool list}}{\Gamma \vdash (\text{fun } f \rightarrow f \ 2 :: [\text{true}]) : (\text{int} \rightarrow \text{bool}) \rightarrow \text{bool list}}$$



(Monomorphic) Let and Let Rec

- let rule:

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

- let rec rule:

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



Example

- Which rule do we apply?

?

| - (let rec one = 1 :: one in

let x = 2 in

fun y -> (x :: y :: one)) : int → int list



Proof of 1

- Which rule?

$[one : \text{int list}] \vdash (1 :: one) : \text{int list}$



Proof of 1

- Application

③

$[one : int\ list] \vdash$
 $((::) 1) : int\ list \rightarrow int\ list$

④

$[one : int\ list] \vdash$
 $one : int\ list$

$[one : int\ list] \vdash (1 :: one) : int\ list$



Proof of 3

Constants Rule

Constants Rule

$[one : int\ list] \vdash$

$((::) : int \rightarrow int\ list \rightarrow int\ list)$

$[one : int\ list] \vdash$

$1 : int$

$[one : int\ list] \vdash ((::) 1) : int\ list \rightarrow int\ list$



Proof of 4

- Rule for variables

$$\frac{}{[one : int\ list] \vdash one:int\ list}$$



Proof of 2

- Constant

⑤ $[x:\text{int}; \text{one} : \text{int list}] \vdash$
 $\text{fun } y \rightarrow$
 $(x :: y :: \text{one}))$

$[\text{one} : \text{int list}] \vdash 2:\text{int} \quad : \text{int} \rightarrow \text{int list}$

$[\text{one} : \text{int list}] \vdash (\text{let } x = 2 \text{ in}$
 $\text{fun } y \rightarrow (x :: y :: \text{one})) : \text{int} \rightarrow \text{int list}$



Proof of 5

?

$[x:\text{int}; \text{one} : \text{int list}] \vdash \text{fun } y \rightarrow (x :: y :: \text{one}))$
 $: \text{int} \rightarrow \text{int list}$



Proof of 5

?

$$[y:\text{int}; x:\text{int}; \text{one} : \text{int list}] \vdash (x :: y :: \text{one}) : \text{int list}$$

$$[x:\text{int}; \text{one} : \text{int list}] \vdash \text{fun } y \text{ -> } (x :: y :: \text{one})) \\ : \text{int} \rightarrow \text{int list}$$



Proof of 5

⑥

$[y:\text{int}; x:\text{int}; \text{one} : \text{int list}] \vdash [y:\text{int}; x:\text{int}; \text{one} : \text{int list}] \vdash$

$((::) x) : \text{int list} \rightarrow \text{int list}$

⑦

$(y :: \text{one}) : \text{int list}$

$[y:\text{int}; x:\text{int}; \text{one} : \text{int list}] \vdash (x :: y :: \text{one}) : \text{int list}$

$[x:\text{int}; \text{one} : \text{int list}] \vdash \text{fun } y \text{ -> } (x :: y :: \text{one}) : \text{int} \rightarrow \text{int list}$



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

•
•
•

$$\frac{[y:\text{int}; \dots] \vdash ((::) y) \quad \vdash \text{int list} \rightarrow \text{int list}}{[y:\text{int}; x:\text{int}; \text{one} : \text{int list}] \vdash (y :: \text{one}) : \text{int list}}
 \quad
 \frac{[\dots; \text{one} : \text{int list}] \vdash \quad \vdash \text{one} : \text{int list}}{[\dots; \text{one} : \text{int list}] \vdash (y :: \text{one}) : \text{int list}}$$



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

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



Remaining Problems

- 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

- Monomorphic Types (τ):
 - Basic Types: int, bool, float, string, unit, ...
 - Type Variables: $\alpha, \beta, \gamma, \delta, \varepsilon$
 - Compound Types: $\alpha \rightarrow \beta, \text{int} * \text{string}, \text{bool list}, \dots$
- Polymorphic Types:
 - Monomorphic types τ
 - Universally quantified monomorphic types
 - $\forall \alpha_1, \dots, \alpha_n . \tau$
 - 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 $\text{FreeVars}(\tau)$
- Free variables of polymorphic type removes variables that are universally quantified
 - $\text{FreeVars}(\forall \alpha_1, \dots, \alpha_n . \tau) = \text{FreeVars}(\tau) - \{\alpha_1, \dots, \alpha_n\}$
- $\text{FreeVars}(\Gamma) =$ 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
- Recall `a vs. `_a
- $\text{Gen}(\tau, \Gamma) = \forall \alpha_1, \dots, \alpha_n . \tau$ where
 $\{\alpha_1, \dots, \alpha_n\} = \text{freeVars}(\tau) - \text{freeVars}(\Gamma)$



Polymorphic Typing Rules

- A *type judgment* has the form
$$\Gamma \vdash \text{exp} : \tau$$
 - 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:

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

- Where φ replaces all occurrences of $\alpha_1, \dots, \alpha_n$ by monotypes τ_1, \dots, τ_n
- Note: Monomorphic rule special case:

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

- Constants treated same way



Fun Rule Stays the Same

- fun rule:

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

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



Polymorphic Example

- Assume additional constants:
- $\text{hd} : \forall \alpha. \alpha \text{ list} \rightarrow \alpha$
- $\text{tl} : \forall \alpha. \alpha \text{ list} \rightarrow \alpha \text{ list}$
- $\text{is_empty} : \forall \alpha. \alpha \text{ list} \rightarrow \text{bool}$
- $:: : \forall \alpha. \alpha \rightarrow \alpha \text{ list} \rightarrow \alpha \text{ list}$
- $[] : \forall \alpha. \alpha \text{ list}$



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

<p>■ Show: (1)</p> $\{ \text{length} : \alpha \text{ list} \rightarrow \text{int} \}$ $\vdash \text{fun } l \rightarrow \dots$ $: \alpha \text{ list} \rightarrow \text{int}$	<p>(2)</p> $\{ \text{length} : \forall \alpha. \alpha \text{ list} \rightarrow \text{int} \}$ $\vdash \text{length } ((::) 2 []) +$ $\text{length}((::) \text{true } []) : \text{int}$
$\{ \} \vdash \text{let rec length} =$ $\quad \text{fun } l \rightarrow \text{if is_empty } l \text{ then } 0$ $\quad \quad \text{else } 1 + \text{length } (\text{tl } l)$ $\text{in length } ((::) 2 []) + \text{length}((::) \text{true } []) : \text{int}$	



Polymorphic Example (1)

- Show:

?

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



Polymorphic Example (1): Fun Rule

■ Show: (3)

$\{ \text{length} : \alpha \text{ list} \rightarrow \text{int}, \text{ l} : \alpha \text{ list} \} \vdash$

if is_empty l then 0

else length (hd l) + length (tl l) : int

$\{ \text{length} : \alpha \text{ list} \rightarrow \text{int} \} \vdash$

fun l -> if is_empty l then 0

else 1 + length (tl l)

: $\alpha \text{ list} \rightarrow \text{int}$



Polymorphic Example (3)

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

?

$\Gamma \vdash \text{if is_empty } l \text{ then } 0$
 $\text{else } 1 + \text{length (tl } l) : \text{int}$



Polymorphic Example (3): IfThenElse

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

$$\begin{array}{ccc} (4) & (5) & (6) \\ \Gamma \vdash \text{is_empty } l & \Gamma \vdash 0 : \text{int} & \Gamma \vdash 1 + \\ & : \text{bool} & \text{length } (\text{tl } l) : \text{int} \\ \hline \Gamma \vdash \text{if is_empty } l \text{ then } 0 \\ & \text{else } 1 + \text{length } (\text{tl } l) : \text{int} \end{array}$$



Polymorphic Example (4)

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

?

$\Gamma \vdash \text{is_empty l} : \text{bool}$



Polymorphic Example (4): Application

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

?

?

$$\Gamma \vdash \text{is_empty} : \alpha \text{ list} \rightarrow \text{bool}$$

$$\Gamma \vdash \text{l} : \alpha \text{ list}$$

$$\Gamma \vdash \text{is_empty l} : \text{bool}$$



Polymorphic Example (4)

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

By Const since $\alpha \text{ list} \rightarrow \text{bool}$ is
instance of $\forall \alpha. \alpha \text{ list} \rightarrow \text{bool}$?

$$\Gamma \vdash \text{is_empty} : \alpha \text{ list} \rightarrow \text{bool}$$

$$\Gamma \vdash \text{l} : \alpha \text{ list}$$

$$\Gamma \vdash \text{is_empty l} : \text{bool}$$



Polymorphic Example (4)

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

By Const since $\alpha \text{ list} \rightarrow \text{bool}$ is instance of $\forall \alpha. \alpha \text{ list} \rightarrow \text{bool}$ By Variable $\Gamma(l) = \alpha \text{ list}$

$$\Gamma \vdash \text{is_empty} : \alpha \text{ list} \rightarrow \text{bool}$$

$$\Gamma \vdash l : \alpha \text{ list}$$

$$\Gamma \vdash \text{is_empty } l : \text{bool}$$

- This finishes (4)



Polymorphic Example (5):Const

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

- Show

By Const Rule

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

Polymorphic Example (6): Arith Op

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

By Variable

$$\frac{}{\Gamma \vdash \text{length}} \quad (7)$$

By Const

$$\frac{}{1 : \alpha \text{ list} \rightarrow \text{int}} \quad \Gamma \vdash (tl\ l) : \alpha \text{ list}$$

$$\frac{}{\Gamma \vdash 1 : \text{int}}$$

$$\frac{}{\Gamma \vdash \text{length } (tl\ l) : \text{int}}$$

$$\frac{}{\Gamma \vdash 1 + \text{length } (tl\ l) : \text{int}}$$



Polymorphic Example (7):App Rule

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

$$\frac{\begin{array}{c} \text{By Const} \\ \hline \Gamma \vdash \text{tl} : \alpha \text{ list} \rightarrow \alpha \text{ list} \end{array} \quad \begin{array}{c} \text{By Variable} \\ \hline \Gamma \vdash l : \alpha \text{ list} \end{array}}{\Gamma \vdash (\text{tl } l) : \alpha \text{ list}}$$

By Const since $\alpha \text{ list} \rightarrow \alpha \text{ list}$ is instance of
 $\forall \alpha. \alpha \text{ list} \rightarrow \alpha \text{ list}$

Polymorphic Example: (2) by ArithOp

- Let $\Gamma' = \{\text{length} \forall \alpha. \alpha \text{ list} \rightarrow \text{int}\}$
- Show:

(8)

$\Gamma' \vdash$

$\text{length} ((::) 2 []) : \text{int}$

(9)

$\Gamma' \vdash$

$\text{length} ((::) \text{true} []) : \text{int}$

$\{\text{length}: \alpha. \alpha \text{ list} \rightarrow \text{int}\}$

$\vdash \text{length} ((::) 2 []) + \text{length} ((::) \text{true} []) : \text{int}$



Polymorphic Example: (8)AppRule

- Let $\Gamma' = \{\text{length} \forall \alpha. \alpha \text{ list} \rightarrow \text{int}\}$
- Show:

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



Polymorphic Example: (8)AppRule

- Let $\Gamma' = \{\text{length} \forall \alpha. \alpha \text{ list} \rightarrow \text{int}\}$

- Show:

By Var since $\text{int list} \rightarrow \text{int}$ is instance of

$\forall \alpha. \alpha \text{ list} \rightarrow \text{int}$

(10)

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



Polymorphic Example: (10)AppRule

- Let $\Gamma' = \{\text{length}.\forall \alpha. \alpha \text{ list} \rightarrow \text{int}\}$
- Show:
- By Const since $\alpha \text{ list}$ is instance of $\forall \alpha. \alpha \text{ list}$

(11)

$$\frac{\Gamma' \vdash ((::) 2) : \text{int list} \rightarrow \text{int list} \quad \overline{\Gamma' \vdash [] : \text{int list}}}{\Gamma' \vdash ((::) 2 []) : \text{int list}}$$

Polymorphic Example: (11)AppRule

- Let $\Gamma' = \{\text{length} \forall \alpha. \alpha \text{ list} \rightarrow \text{int}\}$
- Show:
- By Const since $\alpha \text{ list}$ is instance of

$$\frac{\forall \alpha. \alpha \text{ list}}{\Gamma' \vdash ((::) : \text{int} \rightarrow \text{int list} \rightarrow \text{int list}) : \text{int list} \rightarrow \text{int list}} \quad \frac{\text{By Const}}{\Gamma' \vdash 2 : \text{int}}$$

$$\Gamma' \vdash (((::) 2) : \text{int list} \rightarrow \text{int list})$$



Polymorphic Example: (9)AppRule

- Let $\Gamma' = \{\text{length}.\forall \alpha. \alpha \text{ list} \rightarrow \text{int}\}$
- Show:

$$\frac{\Gamma' \vdash \text{length}:\text{bool list} \rightarrow \text{int} \quad \Gamma' \vdash ((::) \text{true } []):\text{bool list}}{\Gamma' \vdash \text{length } ((::) \text{true } []) : \text{int}}$$



Polymorphic Example: (9)AppRule

- Let $\Gamma' = \{\text{length} \forall \alpha. \alpha \text{ list} \rightarrow \text{int}\}$

- Show:

By Var since $\text{bool list} \rightarrow \text{int}$ is instance of

$\forall \alpha. \alpha \text{ list} \rightarrow \text{int}$

(12)

 $\Gamma' \vdash$ $\text{length} : \text{bool list} \rightarrow \text{int}$ $\Gamma' \vdash$ $((::) \text{ true } []): \text{bool list}$

 $\Gamma' \vdash \text{length } ((::) \text{ true } []) : \text{int}$



Polymorphic Example: (12)AppRule

- Let $\Gamma' = \{\text{length} \forall \alpha. \alpha \text{ list} \rightarrow \text{int}\}$
- Show:
- By Const since $\alpha \text{ list}$ is instance of $\forall \alpha. \alpha \text{ list}$

(13)

$$\frac{\Gamma' \vdash ((::)\text{true}): \text{bool list} \rightarrow \text{bool list} \quad \overline{\Gamma' \vdash []: \text{bool list}}}{\Gamma' \vdash ((::)\text{true} []) : \text{bool list}}$$
$$\Gamma' \vdash ((::)\text{true} []) : \text{bool list}$$

Polymorphic Example: (13)AppRule

- Let $\Gamma' = \{\text{length}.\forall \alpha. \alpha \text{ list} \rightarrow \text{int}\}$

- Show:

By Const since bool list

is instance of $\forall \alpha. \alpha \text{ list}$

By Const

$$\frac{\Gamma' \vdash \quad \quad \quad \Gamma' \vdash}{\begin{array}{l} (\::): \text{bool} \rightarrow \text{bool list} \rightarrow \text{bool list} \quad \text{true} : \text{bool} \\ \Gamma' \vdash ((\::) \text{true}) : \text{bool list} \rightarrow \text{bool list} \end{array}}$$