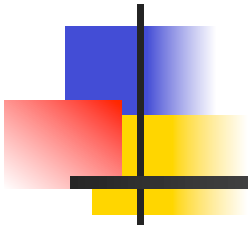


Programming Languages and Compilers (CS 421)



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

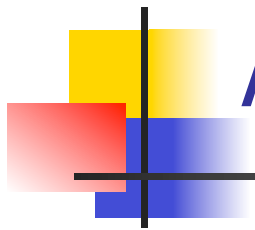


Format of Type Judgments

- 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

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

$\frac{}{\vdash \text{true} : \text{bool}}$

$\frac{}{\vdash \text{false} : \text{bool}}$

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



Axioms – Variables (Monomorphic Rule)

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:

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



Simple Rules - Arithmetic

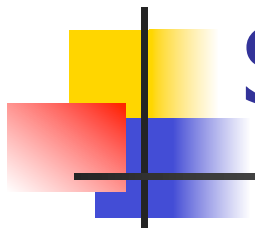
Primitive operators ($\oplus \in \{ +, -, *, \dots \}$):

$$\frac{\Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau \quad (\oplus) : \tau \rightarrow \tau \rightarrow \tau}{\Gamma \vdash e_1 \oplus e_2 : \tau}$$

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



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

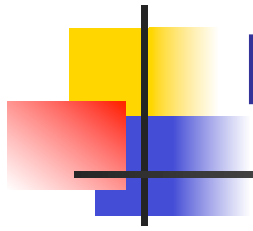
- Which rule do we apply?

?

| - (let rec one = 1 :: one in
 let x = 2 in
 fun y -> (x :: y :: one)) : int \rightarrow int list



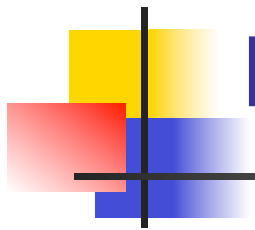
13



Proof of 1

- Which rule?

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



Proof of 1

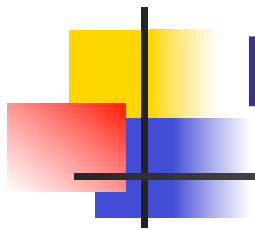
■ Application

③

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

④

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



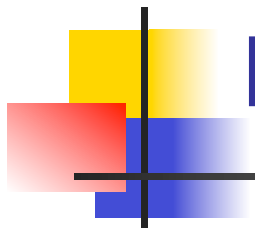
Proof of 3

Constants Rule

$$\frac{[one : int\ list] \vdash \quad (::) : int \rightarrow int\ list \rightarrow int\ list}{[one : int\ list] \vdash ((::) 1) : int\ list \rightarrow int\ list}$$

Constants Rule

$$\frac{[one : int\ list] \vdash \quad 1 : int}{[one : int\ list] \vdash ((::) 1) : int\ list \rightarrow int\ list}$$



Proof of 4

- Rule for variables

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



Proof of 2

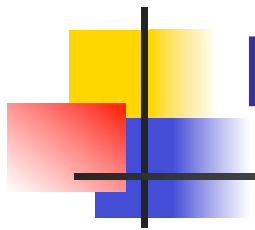
⑤ $[x:\text{int}; \text{one} : \text{int list}] \vdash$
 $\text{fun } y \rightarrow$

■ Constant

$(x :: y :: \text{one}))$

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

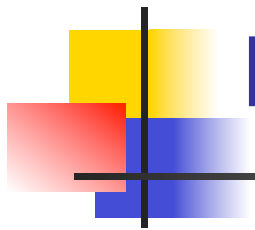
$[\text{one} : \text{int list}] \vdash (\text{let } x = 2 \text{ in}$
 $\text{fun } y \rightarrow (x :: y :: \text{one})) : \text{int} \rightarrow \text{int list}$



Proof of 5

?

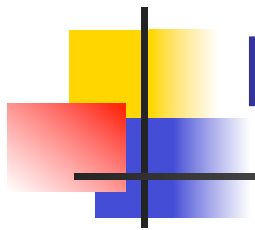
$$[x:\text{int}; \text{one} : \text{int list}] \vdash \text{fun } y \rightarrow (x :: y :: \text{one})) \\ : \text{int} \rightarrow \text{int list}$$



Proof of 5

?

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



Proof of 5

⑥

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

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

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

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

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



Proof of 6

Constant

Variable

$[...] \vdash (::)$

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

$[...; x:\text{int};...] \vdash x:\text{int}$

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

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



Proof of 7

Pf of 6 $[y/x]$

•
•
•

Variable

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

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

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



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

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

- 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