Types

CS 456 - Programming Languages

Errors

- Kind of Errors
  - Checked: Reported
  - Unchecked: Unreported
    (the program maybe continues)
- When
  - Runtime
    (requires testing or proofs for safety)
  - Compile time

Type Errors

- All languages have types
  - What are the types of ImpCore? μScheme?
    - Which are their type errors?
- When do these languages detect type errors?
- What are other kind of errors?

Types

- Types not *only* detect “errors”
  (a form of static analysis)
- It can make programs more efficient
  (don’t need to check preconditions in runtime)
- Different levels of precision
- Not all *safe* programs can be type checked
  (types limit the programs that can be written)

Syntax

$$
\begin{array}{lcl}
exp & ::= & value \\
 & | & variable\text{-}name \\
 & | & (\texttt{set}\ variable\text{-}name\ exp) \\
 & | & (\texttt{if}\ exp\ exp\ exp) \\
 & | & (\texttt{while}\ exp\ exp) \\
 & | & (\texttt{begin}\ \{exp\}) \\
 & | & (function\ \{exp\})
\end{array}
$$

$$
\begin{array}{lcl}
def & ::= & exp \\
 & | & (\texttt{use}\ file\text{-}name) \\
 & | & (\texttt{val}\ variable\text{-}name\ exp) \\
 & | & (\texttt{define}\ function\text{-}name\ (formals)\ exp)
\end{array}
$$

$$
\begin{array}{lcl}
formals & ::= & \{(type\ variable\text{-}name)\} \\
type & ::= & \texttt{int} \mid \texttt{bool} \mid \texttt{unit} \mid (\texttt{array}\ type) \\
value & ::= & integer \\
function & ::= & function\text{-}name \mid primitive \\
primitive & ::= & + \mid - \mid * \mid / \mid = \mid < \mid > \mid \texttt{print}
\end{array}
$$

```
#t = (= 1 1)
#f = (= 1 0)
```

```
(define bool and ((bool x) (bool y)) (if x y x))
```

Types

Primitive types

$$\tau \quad ::= \quad \texttt{INT} \mid \texttt{BOOL} \mid \texttt{UNIT}$$

Type Constructors

$$\mid \quad \texttt{ARRAY}(\tau)$$

$$\mid \quad \tau_1 \times \cdots \times \tau_n \to \tau$$

```
type userfun = { formals : (name * ty) list, body : exp, returns : ty }
```

Type Judgments

$$\Gamma_\xi, \Gamma_\phi, \Gamma_\rho \vdash e : \tau$$

The expression e has type τ under the environments $\Gamma_\xi, \Gamma_\phi, \Gamma_\rho$

$$\Gamma\_ : name \to \tau$$

Records the types of declared variables

Typing Rules

LITERAL

$$\frac{}{\Gamma_\xi, \Gamma_\phi, \Gamma_\rho \vdash \texttt{LITERAL}(v) : \texttt{INT}}$$

FORMAL VAR

$$\frac{x \in \mathsf{dom}(\Gamma_\rho) \qquad \Gamma_\rho(x) = \tau}{\Gamma_\xi, \Gamma_\phi, \Gamma_\rho \vdash \texttt{VAR}(x) : \tau}$$

GOBAL VAR

$$\frac{x \notin \mathsf{dom}(\Gamma_\rho) \qquad x \in \mathsf{dom}(\Gamma_\xi) \qquad \Gamma_\xi(x) = \tau}{\Gamma_\xi, \Gamma_\phi, \Gamma_\rho \vdash \texttt{VAR}(x) : \tau}$$

Typing Rules

FORMAL ASSIGN

$$\frac{x \in \mathsf{dom}(\Gamma_\rho) \qquad \Gamma_\rho(x) = \tau \qquad \Gamma_\xi, \Gamma_\phi, \Gamma_\rho \vdash e : \tau}{\Gamma_\xi, \Gamma_\phi, \Gamma_\rho \vdash \texttt{SET}(x, e) : \tau}$$

GLOBAL ASSIGN

$$\frac{x \notin \mathsf{dom}(\Gamma_\rho) \qquad x \in \mathsf{dom}(\Gamma_\xi) \qquad \Gamma_\xi(x) = \tau \qquad \Gamma_\xi, \Gamma_\phi, \Gamma_\rho \vdash e : \tau}{\Gamma_\xi, \Gamma_\phi, \Gamma_\rho \vdash \texttt{SET}(x, e) : \tau}$$

Typing Rules

IF

$$\frac{\Gamma_\xi, \Gamma_\phi, \Gamma_\rho \vdash e : \texttt{BOOL} \qquad \Gamma_\xi, \Gamma_\phi, \Gamma_\rho \vdash e_0 : \tau \qquad \Gamma_\xi, \Gamma_\phi, \Gamma_\rho \vdash e_1 : \tau}{\Gamma_\xi, \Gamma_\phi, \Gamma_\rho \vdash \texttt{IF}(e, e_0, e_1) : \tau}$$

WHILE

$$\frac{\Gamma_\xi, \Gamma_\phi, \Gamma_\rho \vdash e_0 : \texttt{BOOL} \qquad \Gamma_\xi, \Gamma_\phi, \Gamma_\rho \vdash e_1 : \tau}{\Gamma_\xi, \Gamma_\phi, \Gamma_\rho \vdash \texttt{WHILE}(e_0, e_1) : \texttt{UNIT}}$$

Typing Rules

Empty Begin

$$\Gamma_\xi, \Gamma_\phi, \Gamma_\rho \vdash \texttt{BEGIN}() : \texttt{UNIT}$$

Begin

$$\frac{\Gamma_\xi, \Gamma_\phi, \Gamma_\rho \vdash e_1 : \tau_1 \qquad \ldots \qquad \Gamma_\xi, \Gamma_\phi, \Gamma_\rho \vdash e_n : \tau_n}{\Gamma_\xi, \Gamma_\phi, \Gamma_\rho \vdash \texttt{BEGIN}(e_1, \ldots, e_n) : \tau_n}$$

Typing Rules

APPLY

$$\frac{f \in \mathsf{dom}(\Gamma_\phi) \qquad \Gamma_\phi(f) = \tau_1 \times \cdots \times \tau_n \to \tau \qquad \Gamma_\xi, \Gamma_\phi, \Gamma_\rho \vdash e_i : \tau_i}{\Gamma_\xi, \Gamma_\phi, \Gamma_\rho \vdash \texttt{APPLY}(f, e_1, \ldots, e_n) : \tau}$$

Typing Rules

Apply Eq

$$\frac{\Gamma_\xi, \Gamma_\phi, \Gamma_\rho \vdash e_0 : \tau \qquad \Gamma_\xi, \Gamma_\phi, \Gamma_\rho \vdash e_1 : \tau}{\Gamma_\xi, \Gamma_\phi, \Gamma_\rho \vdash \texttt{APPLY}(=, e_0, e_1) : \texttt{BOOL}}$$

Apply Print

$$\frac{\Gamma_\xi, \Gamma_\phi, \Gamma_\rho \vdash e : \tau}{\Gamma_\xi, \Gamma_\phi, \Gamma_\rho \vdash \texttt{APPLY}(\mathsf{print}, e) : \texttt{UNIT}}$$

Typing Rules

Type Elaboration: when evaluating definitions we collect typing information to type subsequent definitions

$$\langle d, \Gamma_\xi, \Gamma_\phi \rangle \rightarrow \langle \Gamma'_\xi, \Gamma'_\phi \rangle$$

The new environments could have more information

Typing Rules

VAL

$$\frac{\Gamma_\xi, \Gamma_\phi, \{\} \vdash e : \tau}{\langle \mathsf{VAL}(x, e), \Gamma_\xi, \Gamma_\phi \rangle \to \langle \Gamma_\xi\{x \mapsto \tau\}, \Gamma_\phi \rangle}$$

EXP

$$\frac{\langle \mathtt{VAL}(\mathtt{it}, e), \Gamma_\xi, \Gamma_\phi \rangle \to \langle \Gamma'_\xi, \Gamma_\phi \rangle}{\langle \mathtt{EXP}(e), \Gamma_\xi, \Gamma_\phi \rangle \to \langle \Gamma'_\xi, \Gamma_\phi \rangle}$$

Typing Rules

BASE

$$\frac{\tau \in \{\texttt{UNIT}, \texttt{INT}, \texttt{BOOL}\}}{\tau \text{ is a type}}$$

DEFINE

$$\frac{\begin{array}{c}\tau_1, \ldots, \tau_n \text{ are types} \\ \Gamma_\xi, \Gamma_\phi\{f \mapsto \tau_1 \times \cdots \times \tau_n \to \tau\}, \{x_1 \mapsto \tau_1, \ldots, x_n \mapsto \tau_n\} \vdash e : \tau\end{array}}{\langle \texttt{DEFINE}(f, \langle x_1 : \tau_1, \ldots, x_n : \tau_n \rangle), \Gamma_\xi, \Gamma_\phi \rangle \to \langle \Gamma_\xi, \Gamma_\phi\{f \mapsto \tau_1 \times \cdots \times \tau_n \to \tau\} \rangle}$$

Arrays

```
(array-make e1 e2)
 (array-get e1 e2)
(array-set e1 e2)
  (array-legth e)
```

$$
\begin{array}{rcl}
exp & ::= & \texttt{AGET}\ (exp, exp) \\
 & | & \texttt{ASET}\ (exp, exp, exp) \\
 & | & \texttt{AMAKE}\ (exp, exp) \\
 & | & \texttt{ALEN}\ exp
\end{array}
$$

Example code

Array Typing Rules

$$\textsc{Array Formation}\ \frac{\tau \text{ is a type}}{\texttt{ARRAY}(\tau) \text{ is a type}}$$

$$\textsc{Array Make}\ \frac{\Gamma_\xi, \Gamma_\phi, \Gamma_\rho \vdash e_1 : \texttt{INT} \qquad \Gamma_\xi, \Gamma_\phi, \Gamma_\rho \vdash e_2 : \tau}{\Gamma_\xi, \Gamma_\phi, \Gamma_\rho \vdash \texttt{ARRAY} - \texttt{MAKE}(e_1, e_2) : \texttt{ARRAY}(\tau)}$$

$$\textsc{Array Get}\ \frac{\Gamma_\xi, \Gamma_\phi, \Gamma_\rho \vdash e_1 : \texttt{ARRAY}(\tau) \qquad \Gamma_\xi, \Gamma_\phi, \Gamma_\rho \vdash e_2 : \texttt{INT}}{\Gamma_\xi, \Gamma_\phi, \Gamma_\rho \vdash \texttt{ARRAY} - \texttt{GET}(e_1, e_2) : \tau}$$

Array Typing Rules

Array Set

$$\frac{\Gamma_\xi, \Gamma_\phi, \Gamma_\rho \vdash e_1 : \texttt{ARRAY}(\tau) \qquad \Gamma_\xi, \Gamma_\phi, \Gamma_\rho \vdash e_2 : \texttt{INT} \qquad \Gamma_\xi, \Gamma_\phi, \Gamma_\rho \vdash e_3 : \tau}{\Gamma_\xi, \Gamma_\phi, \Gamma_\rho \vdash \texttt{ARRAY} - \texttt{SET}(e_1, e_2, e_3) : \tau}$$

Array Length

$$\frac{\Gamma_\xi, \Gamma_\phi, \Gamma_\rho \vdash e : \texttt{ARRAY}(\tau)}{\Gamma_\xi, \Gamma_\phi, \Gamma_\rho \vdash \texttt{ARRAY} - \texttt{LEN}(e) : \texttt{INT}}$$

Other Common Types

Other Types (Functions)

Arrow Formation

$$\frac{\tau_1, \tau_2 \text{ are types}}{\tau_1 \to \tau_2 \text{ is a type}}$$

Arrow Introduction

$$\frac{\Gamma\{x \to \tau\} \vdash e : \tau'}{\Gamma \vdash \texttt{LAMBDA}(x : \tau, e) : \tau \to \tau'}$$

Arrow Elimination

$$\frac{\Gamma \vdash e_1 : \tau \to \tau' \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \texttt{APPLY}(e_1, e_2) : \tau'}$$

Other Types (Pairs)

Pair Formation

$$\frac{\tau_1, \tau_2 \text{ are types}}{\tau_1 \times \tau_2 \text{ is a type}}$$

Pair Introduction

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \mathsf{PAIR}(e_1, e_2) : \tau_1 \times \tau_2}$$

Pair Elimination (Fst)

$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \mathsf{FST}(e) : \tau_1}$$

Pair Elimination (Snd)

$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \mathsf{SND}(e) : \tau_2}$$

Other Types (Sums)

Sum Formation

$$\frac{\tau_1, \tau_2 \text{ are types}}{\tau_1 + \tau_2 \text{ is a type}}$$

Sum Introduction (Left)

$$\frac{\Gamma \vdash e : \tau_1 \qquad \tau_2 \text{ is a type}}{\Gamma \vdash \texttt{LEFT}_{\tau_2}(e) : \tau_1 \times \tau_2}$$

Sum Introduction (Right)

$$\frac{\Gamma \vdash e : \tau_2 \qquad \tau_1 \text{ is a type}}{\Gamma \vdash \texttt{RIGHT}_{\tau_1}(e) : \tau_1 \times \tau_2}$$

Sum Elimination

$$\frac{\begin{array}{c}\Gamma \vdash e : \tau_1 + \tau_2 \\ \Gamma\{x_1 \mapsto \tau_1\} \vdash e_1 : \tau \qquad \Gamma\{x_2 \mapsto \tau_2\} \vdash e_2 : \tau\end{array}}{\Gamma \vdash \big(\mathsf{case}\ e\ \mathsf{of}\ \texttt{LEFT}(x_1) \Rightarrow e_1 \mid \texttt{RIGHT}(x_2) \Rightarrow e_2\big) : \tau_2 \times \tau}$$

Typed μScheme

Typed μScheme Syntax

$$
\begin{array}{rcl}
def & ::= & exp \\
& | & (\texttt{use}\ file\text{-}name) \\
& | & (\texttt{val}\ variable\text{-}name\ exp) \\
& | & (\texttt{val-rec}\ type\text{-}exp\ variable\text{-}name\ exp) \\
& | & (\texttt{define}\ function\text{-}name\ (formals)\ exp)
\end{array}
$$

$$
\begin{array}{rcl}
exp & ::= & literal \\
& | & variable\text{-}name \\
& | & (\texttt{set}\ variable\text{-}name\ exp) \\
& | & (\texttt{if}\ exp\ exp\ exp) \\
& | & (\texttt{while}\ exp\ exp) \\
& | & (\texttt{begin}\ \{exp\}) \\
& | & (exp\ \{exp\}) \\
& | & (let\text{-}keyword\ (\{(variable\text{-}name\ exp)\})\ exp) \\
& | & (\texttt{lambda}\ (formals)\ exp) \\
& | & (\texttt{type-lambda}\ (type\text{-}formals)\ exp) \\
& | & (@\ exp\ \{type\text{-}exp\})
\end{array}
$$

Typed μScheme Syntax

$$
\begin{array}{lcl}
\textit{let-keyword} & ::= & \texttt{let} \mid \texttt{let*} \\
\textit{formals} & ::= & \{(\textit{type-exp}\ \textit{variable-name})\} \\
\textit{type-formals} & ::= & \{'\textit{type-variable-name}\} \\
\textit{type-exp} & ::= & \textit{type-constructor-name} \\
 & \mid & '\textit{type-variable-name} \\
 & \mid & (\texttt{forall}\ (\{'\textit{type-variable-name}\})\ \textit{type-exp}) \\
 & \mid & (\texttt{function}\ (\{\textit{type-exp}\})\ \textit{type-exp}) \\
 & \mid & (\textit{type-exp}\ \{\textit{type-exp}\}) \\
\textit{literal} & ::= & \textit{integer} \mid \texttt{\#t} \mid \texttt{\#f} \mid '\textit{S-exp} \mid (\texttt{quote}\ \textit{S-exp}) \\
\textit{S-exp} & ::= & \textit{literal} \mid \textit{symbol-name} \mid (\{\textit{S-exp}\})
\end{array}
$$

Kinds (The types of types)

$$\kappa \quad ::= \quad * \mid \kappa_1 \times \kappa_2 \times \cdots \times \kappa_n \Rightarrow \kappa$$

Kind Formation Type

$$\frac{}{* \text{ is a kind}}$$

Kind Formation Arrow

$$\frac{\kappa_1, \kappa_n, \kappa \text{ are kinds}}{\kappa_1 \times \cdots \times \kappa_n \Rightarrow \kappa \text{ is a kind}}$$

$$\begin{aligned}\Delta_0 \quad = \{ \quad & int :: *,\ bool :: *,\ unit :: *, \\ & pair :: * \times * \Rightarrow *,\ sum :: * \times * \Rightarrow *,\ \rightarrow :: * \times * \Rightarrow *, \\ & array :: * \Rightarrow *, list :: * \Rightarrow * \quad \}\end{aligned}$$

Kinding environment

Kinds (The types of types)

Kinding environment:
kind of type constructors

$$\Delta \vdash \tau :: \kappa$$

Type τ has kind κ

$\texttt{TYCON}(\mu)$	Type constructor μ
$\texttt{CONAPP}(\tau, [\tau_1, \ldots, \tau_n])$	Type constructor application

Kinding

Kind Constructor

$$\frac{\mu \in \Delta}{\Delta \vdash \texttt{TYCON}(\mu) :: \Delta(\mu)}$$

Kind Application

$$\frac{\Delta \vdash \tau :: \kappa_1 \times \cdots \times \kappa_n \Rightarrow \kappa \qquad \Delta \vdash \tau_i :: \kappa_i}{\Delta \vdash \texttt{CONAPP}(\tau, [\tau_1, \ldots, \tau_n]) :: \kappa}$$

Kind Tuple

$$\frac{\Delta \vdash \tau_i :: *}{\Delta \vdash \texttt{CONAPP}(\texttt{TYCON}(\texttt{tuple}), [\tau_1, \ldots, \tau_n]) :: *}$$

$$\texttt{CONAPP}(\texttt{TYCON}(\texttt{tuple}), [\tau_1, \ldots, \tau_n]) \equiv \tau_1 \times \cdots \times \tau_n$$

Polymorphic Types

```
(define length (xs)
  (if (null? xs) 0 (+ 1 (length (cdr xs)))))

(length '(1 2 3 4))    (length '(#t #t #f))
```

```
(define lengthI int ((list int) xs)
  (if (null? xs) 0 (+ 1 (length (cdr xs)))))

(define lengthB int ((list bool) xs)
  (if (null? xs) 0 (+ 1 (length (cdr xs)))))

...
```

Polymorphic Types

```
(define lengthB int ((list 'a) xs)
  (if (null? xs) 0 (+ 1 (length (cdr xs)))))

 (forall ('a) (function ((list 'a)) int))
```

$=$

$(\forall\alpha.\ \alpha\ \texttt{list} \rightarrow \texttt{int})$

```
((@ length bool) '(#t #f #t #t))
((@ length int) '(1 2 1 3))
((@ length (list bool)) '((#t #f) (#f #t #t) (#t)))
((@ length (list bool)) '((#t #f) (#f #t #t) (1)))
```

Code samples

Polymorphic Types

$\texttt{TYVAR}(\alpha)$ $\texttt{FORALL}(\langle\alpha_1,\ldots,\alpha_2\rangle,\tau)$

Kind Variable Introduction

$$\frac{\alpha \in \texttt{dom}(\Delta)}{\Delta \vdash \texttt{TYVAR}(\alpha) :: \Delta(\alpha)}$$

Kind Forall

$$\frac{\Delta\{\alpha_1 :: *,\ldots,\alpha_n :: *\} \vdash \tau :: *}{\Delta \vdash \texttt{FORALL}(\langle\alpha_1,\ldots,\alpha_n\rangle,\tau) :: *}$$

Polymorphic Types

$$
\begin{array}{rcll}
\tau & ::= & \mu & \text{(Type Constructor)} \\
 & | & \alpha & \text{(Type Variable)} \\
 & | & (\tau_1, \ldots, \tau_n)\ \tau & \text{(Type Instantiation)} \\
 & | & \forall \alpha_1, \ldots, \alpha_n.\ \tau & \text{(Type Abstraction)} \\
 & | & \tau_1 \times \cdots \times \tau_n & \text{(Product Type)} \\
 & | & \tau_1 \times \cdots \times \tau_n \to \tau & \text{(Function/Arrow Type)}
\end{array}
$$

Typed μScheme Abs. Syntax

TyExp	::=	TYCON	(Name)
	|	CONAPP	(TyExp, {TyExp3})
	|	FORALL	({Name}, TyExp)
	|	TYVAR	(Name)
Def	::=	VAL	(Name, Exp)
	|	VALREC	(Name, TyExp, Exp)
	|	EXP	(Exp)
	|	DEFINE	(Name, TyExp, Lambda_Exp)
	|	USE	(Name)
Exp	::=	LITERAL	(Value)
	|	VAR	(Name)
	|	SET	(Name, Exp)
	|	IF	(Exp, Exp, Exp)
	|	WHILE	(Exp, Exp)
	|	BEGIN	({Exp})
	|	APPLY	(Name, {Exp})
	|	LET	(LetKeyword, {Name}, {Exp}, Exp)
	|	LAMBDA	(Lambda_Exp)
	|	TYLAMBDA	({Name}, Exp)
	|	TYAPPLY	(Exp, {TyExp})
LetKeyword	::=	Let | Let*	

Typed μScheme Abs. Syntax

$$
\begin{array}{lcll}
\texttt{Value} & ::= & \texttt{NIL} & \\
 & | & \texttt{BOOL} & (\texttt{bool}) \\
 & | & \texttt{NUM} & (\texttt{int}) \\
 & | & \texttt{SYM} & (\texttt{Name}) \\
 & | & \texttt{PAIR} & (\texttt{Value}, \texttt{Value}) \\
 & | & \texttt{CLOSURE} & (\texttt{Lambda\_Value}, \texttt{Env}) \\
 & | & \texttt{PRIMITIVE} & (\texttt{Primitive}) \\
 & & & \\
\texttt{Primitive} & ::= & \{\texttt{Value}\} \rightarrow \texttt{Value} & \\
\texttt{Lambda\_Exp} & ::= & (\{(\texttt{Name}, \texttt{TyExp})\}, \texttt{Exp}) & \\
\texttt{Lambda\_Value} & ::= & (\{\texttt{Name}\}, \texttt{Exp}) &
\end{array}
$$

Typing (Expressions)

$$\Delta, \Gamma \vdash e : \tau$$

Literal Num

$$\Delta, \Gamma \vdash \texttt{LITERAL}(\texttt{NUM}(n)) : \texttt{int}$$

Literal Bool

$$\Delta, \Gamma \vdash \texttt{LITERAL}(\texttt{BOOL}(b)) : \texttt{bool}$$

Literal Symbol

$$\Delta, \Gamma \vdash \texttt{LITERAL}(\texttt{SYM}(a)) : \texttt{sym}$$

Typing (Expressions)

Literal List Nil

$$\frac{}{\Delta, \Gamma \vdash \texttt{LITERAL}(\texttt{NIL}) : \forall \alpha.\ \alpha\ \texttt{list}}$$

Literal List Singleton

$$\frac{\Delta, \Gamma \vdash \texttt{LITERAL}(\texttt{v}) : \tau}{\Delta, \Gamma \vdash \texttt{LITERAL}(\texttt{PAIR}(\texttt{v}, \texttt{NIL})) : \tau\ \texttt{list}}$$

Literal List Ind

$$\frac{\Delta, \Gamma \vdash \texttt{LITERAL}(\texttt{v}) : \tau \qquad \Delta, \Gamma \vdash \texttt{LITERAL}(\texttt{v}') : \tau\ \texttt{list}}{\Delta, \Gamma \vdash \texttt{LITERAL}(\texttt{PAIR}(\texttt{v}, \texttt{v}')) : \tau\ \texttt{list}}$$

Typing (Expressions)

$$\textsc{Var}\ \frac{x \in \mathsf{dom}(\Gamma) \qquad \Gamma(x) = \tau}{\Delta, \Gamma \vdash \texttt{VAR}(\texttt{x}) : \tau}$$

$$\textsc{Set}\ \frac{x \in \mathsf{dom}(\Gamma) \qquad \Delta, \Gamma \vdash e : \tau \qquad \Gamma(x) = \tau}{\Delta, \Gamma \vdash \texttt{SET}(\texttt{x}, \texttt{e}) : \tau}$$

$$\textsc{While}\ \frac{\Delta, \Gamma \vdash e : \texttt{bool} \qquad \Delta, \Gamma \vdash e' : \tau}{\Delta, \Gamma \vdash \texttt{WHILE}(e, e') : \texttt{unit}}$$

$$\textsc{If}\ \frac{\Delta, \Gamma \vdash e : \texttt{bool} \qquad \Delta, \Gamma \vdash e_0 : \tau \qquad \Delta, \Gamma \vdash e_1 : \tau}{\Delta, \Gamma \vdash \texttt{IF}(e, e_0, e_1) : \texttt{unit}}$$

Typing (Expressions)

Begin Empty

$$\frac{}{\Delta, \Gamma \vdash \texttt{BEGIN}() : \texttt{unit}}$$

Begin Ind

$$\frac{\Delta, \Gamma \vdash e_i : \tau_i}{\Delta, \Gamma \vdash \texttt{BEGIN}(e_1, \ldots, e_n) : \tau_n}$$

Let

$$\frac{\Delta, \Gamma \vdash e_i : \tau_i \qquad \Delta, \Gamma\{x_1 \mapsto \tau_1, \ldots, x_n \mapsto \tau_n\} \vdash e : \tau}{\Delta, \Gamma \vdash \texttt{LET}(\langle x_1, e_1, \ldots, x_n, e_n \rangle, e) : \tau}$$

LetStar Empty

$$\frac{\Delta, \Gamma \vdash e : \tau}{\Delta, \Gamma \vdash \texttt{LETSTAR}(\langle\rangle, e) : \tau}$$

LetStar

$$\frac{\Delta, \Gamma \vdash \texttt{LET}(\langle x_1, e_1 \rangle, \texttt{LETSTAR}(\langle x_2, e_2, \ldots, x_n, e_n \rangle, e)) : \tau}{\Delta, \Gamma \vdash \texttt{LETSTAR}(\langle x_1, e_1, \ldots, x_n, e_n \rangle, e) : \tau}$$

Typing (Expressions)

LAMBDA

$$\frac{\Delta \vdash \tau_i :: * \qquad \Delta, \Gamma\{x_1 \mapsto \tau_1, \ldots x_n \mapsto \tau_n\} \vdash e : \tau}{\Delta, \Gamma \vdash \texttt{LAMBDA}(\langle x_1 : \tau_1, \ldots, x_n : \tau_n \rangle, e) : \tau_1 \times \cdots \times \tau_n \to \tau}$$

APPLY

$$\frac{\Delta, \Gamma \vdash e_i : \tau_i \qquad \Delta, \Gamma \vdash e : \tau_1 \times \cdots \times \tau_n \to \tau}{\Delta, \Gamma \vdash \texttt{APPLY}(e, e_1, \ldots, e_n) : \tau}$$

Typing (Expressions)

TYPE LAMBDA

$$\frac{\Delta\{\alpha_1 :: *, \ldots, \alpha_n :: *, \}, \Gamma \vdash e : \tau}{\Delta, \Gamma \vdash \texttt{TYLAMBDA}(\alpha_1, \ldots, \alpha_n, e) : \forall \alpha_1, \ldots, \alpha_n.\ \tau}$$

TYPE APPLY

$$\frac{\Delta \vdash \tau_i :: * \qquad \Delta, \Gamma \vdash e : \forall \alpha_1, \ldots, \alpha_n.\ \tau}{\Delta, \Gamma \vdash \texttt{TYAPPLY}(e, \tau_1, \ldots, \tau_n) : \tau[\alpha_1 \to \tau_1, \ldots, \alpha_n \to \tau_n]}$$

Typing (Definitions)

Val

$$\frac{\Delta, \Gamma \vdash e : \tau}{\langle \texttt{VAL}(x, e)\Gamma \rangle \to \Gamma\{x \mapsto \tau\}}$$

ValRec

$$\frac{\Delta, \Gamma\{x \mapsto \tau\} \vdash e : \tau}{\langle \texttt{VALREC}(x, \tau, e)\Gamma \rangle \to \Gamma\{x \mapsto \tau\}}$$

Exp

$$\frac{\langle \texttt{VAL}(\texttt{it}, e)\Gamma \rangle \to \Gamma'}{\langle \texttt{EXP}(e), \Gamma \rangle \to \Gamma'}$$

Define

$$\frac{\langle \texttt{VALREC}\big(f, \tau_1 \times \cdots \times \tau_n \to \tau, \texttt{LAMBDA}(\langle x_1 : \tau_1, \ldots, x_n : \tau_n \rangle, e)\big), \Gamma \rangle \to \Gamma'}{\langle \texttt{DEFINE}(f, \tau, \langle x_1 : \tau_1, \ldots, x_n : \tau_n \rangle, e), \Gamma \rangle \to \Gamma'}$$