A Polymorphic Type System for Multi-Staged Exceptions

Hyunjun Eo

Seoul National University

08/04/2006

This is a joint work with In-Sook Kim and Kwangkeun Yi

Outlie

1. Introduction and Examples
2. Operational Semantics
3. Monomorphic Type System (APLAS '2006)
4. Polymorphic Type System
5. Conclusion

Introduction and Examples

Multi-Staged Languages

- Macros, partial evaluation, code generation, etc.
- Normal computation (at stage 0)
  - λ-calculus
- Code composition (at stage > 0)
  - "quasi-quote" in Lisp
    - backquote (‘): create code template
    - comma (`,`): code substitution
    - eval (`eval`): execute code template

```
let x = ‘1
let y = ‘(,x+2)
in eval y
```

Exceptions

- Control diverter
  - Raised exceptions can escape control structures
- Error handler
  - When error occurs, we raise an exception.
  - Then, handlers can catch the raised exception to handle it.
- Possible safety hole
  - Uncaught exceptions cause abnormal termination of programs

```
fun find [] x = raise NotFound
  | find h::t x = if h=x then raise Found
                  else find t x

find [1,2,3] 2 handle Found => true
                    | NotFound => false
```

Exceptions in Multi-Staged Languages

- Restriction
  - exceptions must be raised and handled only at stage 0
- Most interesting feature
  - exceptions raised during code composition
    - can be raised and handled at stage 0
    - can cross stages upwards by comma(,) and downwards by backquote(‘)

Staged Exception Examples (1/4)

```
fun g [] = ‘1
  | g x::r = ‘(,x * ,(g r))

fun f ls = ‘(a * ,(g ls))
```

- input: ‘2::‘0::‘3::[]
- output: ‘(a * 2 * 0 * 3 * 1)

Staged Exception Examples (2/4)

- Raise exception Zero when input has ‘0

```
fun g [] = ‘1
  | g x::r = if x = ‘0 then raise Zero
             else ‘(,x * ,(g r))

fun f ls = ‘(a * ,(g ls))
```

- input: ‘2::‘0::‘3::[]
- output: uncaught exception Zero

Staged Exception Examples (3/4)

- We can handle Zero at stage 0 in the code composition

```
fun g [] = ‘1
  | g x::r = if x = ‘0 then raise Zero
             else ‘(,x * ,(g r))

fun f ls = ‘(a * ,((g ls) handle Zero => ‘0))
```

- input: ‘2::‘0::‘3::[]
- output: ‘(a * 0)

Staged Exception Examples (4/4)

- Or, we can handle `Zero` at stage 0 outside the code composition

```
fun g [] = ‘1
  | g x::r = if x = ‘0 then raise Zero
             else ‘(,x * ,(g r))

fun f ls = ‘(a * ,(g ls))
           handle Zero => ‘0
```

- input: `‘2::‘0::‘3::[]`
- output: `‘0`

Goal

- A static type system that supports
  - Lisp/Scheme's quasi-quote operators and
  - exception facilities

Idea

- effect E: set of possible uncaught exceptions
- expression e has effect E and $c \in E$ means that the expression may raise uncaught exception c.
- code's type: annotated with latent effect

$$\texttt{`(raise } c\texttt{)} : \Box(\Gamma \triangleright A, \{c\}), \emptyset$$

- evaluation of code

$$\texttt{eval `(raise } c\texttt{)} : A, \{c\}$$

Example

```
fun g [] = `1
  | g x::r = if x = `0 then raise Zero
             else `(,x * ,(g r))
```

$$\texttt{g} : \square(\varnothing \triangleright \texttt{int}, \emptyset)\ \texttt{list} \xrightarrow{\{\texttt{Zero}\}} \square(\varnothing \triangleright \texttt{int}, \emptyset), \emptyset$$

```
fun f ls = `(a * ,(g ls))
           handle Zero => `0
```

handler at stage 0

$$\texttt{f} : \square(\varnothing \triangleright \texttt{int}, \emptyset)\ \texttt{list} \xrightarrow{\emptyset} \square(\{\texttt{a} : \texttt{int}\} \triangleright \texttt{int}, \emptyset), \emptyset$$

```
fun f ls = `(a * ,(g ls)
           handle Zero => 0)
```

handler at stage 1

$$\texttt{f} : \square(\varnothing \triangleright \texttt{int}, \emptyset)\ \texttt{list} \xrightarrow{\{\texttt{Zero}\}} \square(\{\texttt{a} : \texttt{int}\} \triangleright \texttt{int}, \emptyset), \emptyset$$

Operational Semantics

Language

e	$\in$	*Exp*	$::=$	$i \mid c \mid x \mid \lambda x.e \mid e_1 e_2$	
			$\mid$	$\texttt{box}\, e$	code template $\quad \text{`}e$
			$\mid$	$\texttt{unbox}_k\, e$	code substitution $\quad ,e$
			$\mid$	$\texttt{eval}\, e$	code execution $\quad \texttt{eval}\, e$
			$\mid$	$\texttt{raise}\, e$	exception raise
			$\mid$	$\texttt{handle}\, e_1\ c\ e_2$	exception handle

- Evaluation

$$e \xrightarrow{n} r$$

where

n: a stage number

r: a value v or raised exception $\overline{c}$

Operational Semantics

- Exceptions must be raised and handled only at stage 0

	Normal computations (at stage 0) and code compositions (at stage $n > 0$)	Propagation of raised exceptions
(ERAISE)	$\dfrac{e \xrightarrow{0} c}{\texttt{raise}\, e \xrightarrow{0} \overline{c}}$	
	$\dfrac{e \xrightarrow{n} v}{\texttt{raise}\, e \xrightarrow{n} \texttt{raise}\, v}(n > 0)$	$\dfrac{e \xrightarrow{n} \overline{c}}{\texttt{raise}\, e \xrightarrow{n} \overline{c}}(n \geq 0)$
(EHANDLE)	$\dfrac{e_1 \xrightarrow{0} v}{\texttt{handle}\, e_1\ c\ e_2 \xrightarrow{0} v}$	$\dfrac{e_1 \xrightarrow{n} \overline{c}}{\texttt{handle}\, e_1\ c\ e_2 \xrightarrow{n} \overline{c}}(n > 0)$
	$\dfrac{e_1 \xrightarrow{0} \overline{c} \quad e_2 \xrightarrow{0} v}{\texttt{handle}\, e_1\ c\ e_2 \xrightarrow{0} v}$	$\dfrac{e_1 \xrightarrow{n} \overline{c}}{\texttt{handle}\, e_1\ c'\ e_2 \xrightarrow{n} \overline{c}}(n > 0)$
	$\dfrac{e_1 \xrightarrow{0} \overline{c}}{\texttt{handle}\, e_1\ c'\ e_2 \xrightarrow{0} \overline{c}}$	$\dfrac{e_2 \xrightarrow{n} \overline{c}}{\texttt{handle}\, e_1\ c\ e_2 \xrightarrow{n} \overline{c}}(n > 0)$

Operational Semantics

- Exceptions can cross stages upwards or downwards

	Normal computations (at stage 0) and code compositions (at stage $n > 0$)	Propagation of raised exceptions
(EBOX)	$\dfrac{e \xrightarrow{n+1} v}{\texttt{box}\, e \xrightarrow{n} \texttt{box}\, v}(n \geq 0)$	$\dfrac{e \xrightarrow{n+1} \overline{c}}{\texttt{box}\, e \xrightarrow{n} \overline{c}}(n \geq 0)$
(EUNBOX)	$\dfrac{e \xrightarrow{0} \texttt{box}\, v}{\texttt{unbox}_k\, e \xrightarrow{n} v}(n = k > 0)$	
	$\dfrac{e \xrightarrow{n-k} v}{\texttt{unbox}_k\, e \xrightarrow{n} \texttt{unbox}_k\, v}(n > k > 0)$	$\dfrac{e \xrightarrow{n-k} \overline{c}}{\texttt{unbox}_k\, e \xrightarrow{n} \overline{c}}(n \geq k > 0)$
(EEVAL)	$\dfrac{e \xrightarrow{0} \texttt{box}\, v^1 \quad v^1 \xrightarrow{0} v^0}{\texttt{eval}\, e \xrightarrow{0} v^0}$	
	$\dfrac{e \xrightarrow{n} v}{\texttt{eval}\, e \xrightarrow{n} \texttt{eval}\, v}(n > 0)$	$\dfrac{e \xrightarrow{n} \overline{c}}{\texttt{eval}\, e \xrightarrow{n} \overline{c}}(n \geq 0)$

Monomorphic Type System

Type and effect

$$A, B \in \mathit{Type} ::= \texttt{int} \mid \texttt{exn}(E) \mid A \xrightarrow{E} B \mid \Box(\Gamma \rhd A, E)$$

$$E \in \mathit{Effects} = 2^{\mathit{Exn}}$$

$$c \in \mathit{Exn} = \text{set of exception names}$$

Typing judgment

$$\Gamma_0 \cdots \Gamma_n \vdash e : A, E_n \cdots E_0$$

Typing rules

$$\frac{c \in E}{\Gamma_0 \cdots \Gamma_n \vdash c : \mathtt{exn}(E), \emptyset E_{n-1} \cdots E_0} \quad \text{(TEXN)}$$

$$\frac{\Gamma_0 \cdots \Gamma_n \vdash e : \mathtt{exn}(E), E_n E_{n-1} \cdots E_0}{\Gamma_0 \cdots \Gamma_n \vdash \mathtt{raise}\, e : A, (E \cup E_n) E_{n-1} \cdots E_0} \quad \text{(TRAISE)}$$

$$\frac{\begin{array}{c}\Gamma_0 \cdots \Gamma_n \vdash e_1 : A, E_n E_{n-1} \cdots E_0 \\ \Gamma_0 \cdots \Gamma_n \vdash e_2 : A, E'_n E'_{n-1} \cdots E'_0\end{array}}{\Gamma_0 \cdots \Gamma_n \vdash \mathtt{handle}\, e_1\ c\ e_2 : A, ((E_n \setminus \{c\}) \cup E'_n)(E_{n-1} \cup E'_{n-1}) \cdots (E_0 \cup E'_0)} \quad \text{(THANDLE)}$$

Typing rules

$$\frac{\Gamma_0 \cdots \Gamma_n \Gamma \vdash e : A, E E_n \cdots E_0}{\Gamma_0 \cdots \Gamma_n \vdash \texttt{box}\, e : \Box(\Gamma \rhd A, E), E_n \cdots E_0} \quad \texttt{(TBOX)}$$

$$\frac{\Gamma_0 \cdots \Gamma_n \vdash e : \Box(\Gamma_{n+k} \rhd A, E), E_n \cdots E_0}{\Gamma_0 \cdots \Gamma_n \cdots \Gamma_{n+k} \vdash \texttt{unbox}_k\, e : A, E\emptyset^{k-1} E_n \cdots E_0} \quad \texttt{(TUNBOX)}$$

$$\frac{\Gamma_0 \cdots \Gamma_n \vdash e : \Box(\varnothing \rhd A, E), E_n E_{n-1} \cdots E_0}{\Gamma_0 \cdots \Gamma_n \vdash \texttt{eval}\, e : A, (E \cup E_n) E_{n-1} \cdots E_0} \quad \texttt{(TEVAL)}$$

Typing rules

$$\frac{\Gamma_0 \cdots \Gamma_n + x : A \vdash e : B, E_n E_{n-1} \cdots E_0}{\Gamma_0 \cdots \Gamma_n \vdash \lambda x.e : A \xrightarrow{E_n} B, \emptyset E_{n-1} \cdots E_0} \quad \texttt{(TABS)}$$

$$\frac{\begin{array}{c}\Gamma_0 \cdots \Gamma_n \vdash e_1 : A \xrightarrow{E} B, E_n E_{n-1} \cdots E_0 \\ \Gamma_0 \cdots \Gamma_n \vdash e_2 : A, E'_n E'_{n-1} \cdots E'_0\end{array}}{\Gamma_0 \cdots \Gamma_n \vdash e_1\ e_2 : B, (E \cup E_n \cup E'_n)(E_{n-1} \cup E'_{n-1}) \cdots (E_0 \cup E'_0)} \quad \texttt{(TAPP)}$$

$$\frac{\Gamma_0 \cdots \Gamma_n \vdash e : A, E_n \cdots E_0 \quad E_n \subseteq E'_n \cdots E_0 \subseteq E'_0}{\Gamma_0 \cdots \Gamma_n \vdash e : A, E'_n \cdots E'_0} \quad \texttt{(TSUB)}$$

Example

```
fun g [] = `1
  | g x::r = if x = `0 then raise Zero
             else `(,x * ,(g r))
```

$$g : \square(\varnothing \triangleright \texttt{int}, \emptyset)\ \texttt{list} \xrightarrow{\{\texttt{Zero}\}} \square(\varnothing \triangleright \texttt{int}, \emptyset), \emptyset$$

```
fun f ls = `(a * ,(g ls))
           handle Zero => `0
```

Example

```
fun g [] = `1
  | g x::r = if x = `0 then raise Zero
             else `(,x * ,(g r))
```

$$\texttt{g} : \square(\varnothing \triangleright \texttt{int}, \emptyset)\ \texttt{list} \xrightarrow{\{\texttt{Zero}\}} \square(\varnothing \triangleright \texttt{int}, \emptyset), \emptyset$$

```
fun f ls = `(a * ,(g ls))
           handle Zero => `0
```

$$\square(\varnothing \triangleright \texttt{int}, \emptyset), \{\texttt{Zero}\}$$

Example

```
fun g [] = `1
  | g x::r = if x = `0 then raise Zero
             else `(,x * ,(g r))
```

$$\texttt{g} : \square(\varnothing \triangleright \texttt{int}, \emptyset)\ \texttt{list} \xrightarrow{\{\texttt{Zero}\}} \square(\varnothing \triangleright \texttt{int}, \emptyset), \emptyset$$

```
fun f ls = `(a * ,(g ls))
           handle Zero => `0
```

$$\texttt{int}, \{\texttt{Zero}\}\emptyset$$

Example

```
fun g [] = `1
  | g x::r = if x = `0 then raise Zero
             else `(,x * ,(g r))
```

$$\texttt{g} : \square(\varnothing \triangleright \texttt{int}, \emptyset)\ \texttt{list} \xrightarrow{\{\texttt{Zero}\}} \square(\varnothing \triangleright \texttt{int}, \emptyset), \emptyset$$

```
fun f ls = `(a * ,(g ls))
           handle Zero => `0
```

$$\square(\{\texttt{a} : \texttt{int}\} \triangleright \texttt{int}, \emptyset), \{\texttt{Zero}\}$$

Example

```
fun g [] = `1
  | g x::r = if x = `0 then raise Zero
             else `(,x * ,(g r))
```

$$g : \square(\varnothing \triangleright \texttt{int}, \emptyset)\ \texttt{list} \xrightarrow{\{\texttt{Zero}\}} \square(\varnothing \triangleright \texttt{int}, \emptyset), \emptyset$$

```
fun f ls = `(a * ,(g ls))
           handle Zero => `0
```

$$\square(\{\texttt{a} : \texttt{int}\} \triangleright \texttt{int}, \emptyset), \emptyset$$

Example

```
fun g [] = `1
  | g x::r = if x = `0 then raise Zero
             else `(,x * ,(g r))
```

$$\texttt{g} : \square(\varnothing \triangleright \texttt{int}, \emptyset)\ \texttt{list} \xrightarrow{\{\texttt{Zero}\}} \square(\varnothing \triangleright \texttt{int}, \emptyset), \emptyset$$

```
fun f ls = `(a * ,(g ls))
           handle Zero => `0
```

$$\texttt{f} : \square(\varnothing \triangleright \texttt{int}, \emptyset)\ \texttt{list} \xrightarrow{\emptyset} \square(\{\texttt{a} : \texttt{int}\} \triangleright \texttt{int}, \emptyset), \emptyset$$

Example

```
fun g [] = `1
  | g x::r = if x = `0 then raise Zero
             else `(,x * ,(g r))
```

$$\texttt{g} : \square(\varnothing \triangleright \texttt{int}, \emptyset)\ \texttt{list} \xrightarrow{\{\texttt{Zero}\}} \square(\varnothing \triangleright \texttt{int}, \emptyset), \emptyset$$

```
fun f ls = `(a * ,(g ls))
           handle Zero => `0
```

handler at stage 0

$$\texttt{int}, \{\texttt{Zero}\}\emptyset$$

```
fun f ls = `(a * ,(g ls)
           handle Zero => 0)
```

handler at stage 1

$$\texttt{int}, \{\texttt{Zero}\}\emptyset$$

Example

```
fun g [] = `1
  | g x::r = if x = `0 then raise Zero
             else `(,x * ,(g r))
```

$$g : \square(\varnothing \triangleright \texttt{int}, \emptyset)\ \texttt{list} \xrightarrow{\{\texttt{Zero}\}} \square(\varnothing \triangleright \texttt{int}, \emptyset), \emptyset$$

```
fun f ls = `(a * ,(g ls))
           handle Zero => `0
```

handler at stage 0

$$\square(\{\texttt{a} : \texttt{int}\} \triangleright \texttt{int}, \emptyset), \{\texttt{Zero}\}$$

```
fun f ls = `(a * ,(g ls)
           handle Zero => 0)
```

handler at stage 1

$$\texttt{int}, \{\texttt{Zero}\}\emptyset$$

Example

```
fun g [] = `1
  | g x::r = if x = `0 then raise Zero
             else `(,x * ,(g r))
```

$$\texttt{g} : \square(\varnothing \triangleright \texttt{int}, \emptyset)\ \texttt{list} \xrightarrow{\{\texttt{Zero}\}} \square(\varnothing \triangleright \texttt{int}, \emptyset), \emptyset$$

```
fun f ls = `(a * ,(g ls))
           handle Zero => `0
```

handler at stage 0

$$\square(\{\texttt{a} : \texttt{int}\} \triangleright \texttt{int}, \emptyset), \emptyset$$

```
fun f ls = `(a * ,(g ls)
           handle Zero => 0)
```

handler at stage 1

$$\square(\{\texttt{a} : \texttt{int}\} \triangleright \texttt{int}, \emptyset), \{\texttt{Zero}\}$$

Soundness

Lemma (Demotion and Promotion)

Suppose $\varnothing\Gamma_1 \cdots \Gamma_n \vdash v : A, E_n \cdots E_0$.

1. *If* $\Gamma_1 = \varnothing$ *then* $\Gamma_1 \cdots \Gamma_n \vdash v : A, E_n \cdots E_1$.
2. *For all* $\Gamma'_1 \cdots \Gamma'_m$, $E'_m \cdots E'_1$,
   $\varnothing\Gamma'_1 \cdots \Gamma'_m\Gamma_1 \cdots \Gamma_n \vdash v : A, E_n \cdots E_1 E'_m \cdots E'_1 E_0$.

Lemma (Empty Effect of v^0)

If $\Gamma_0 \vdash v : A, E$ *then* $\Gamma_0 \vdash v : A, \emptyset$.

Theorem (Soundness)

Suppose $\varnothing\Gamma_1 \cdots \Gamma_n \vdash e : A, E_n \cdots E_0$.

1. *If* $e \xrightarrow{n} v$ *then* $\varnothing\Gamma_1 \cdots \Gamma_n \vdash v : A, E_n \cdots E_0$.
2. *If* $e \xrightarrow{n} \overline{c}$ *then* $E_n \supseteq \{c\}$.

Polymorphic Type System

- What's the type of the following function?
  - $$\lambda x.\texttt{box}\,(\texttt{unbox}_1\,(\texttt{raise}\,x))$$
  - $$\begin{gathered}\texttt{exn}(\{c\}) \xrightarrow{\{c\}} \Box(\varnothing \rhd \texttt{int}, \emptyset) \\ \texttt{exn}(\{c'\}) \xrightarrow{\{c'\}} \Box(\varnothing \rhd \texttt{exn}(\{c\}), \{c''\}) \\ \texttt{exn}(\{c, c'\}) \xrightarrow{\{c,c'\}} \Box(\varnothing \rhd \texttt{int} \xrightarrow{\{c''\}} \texttt{int}, \emptyset) \\ \cdots\end{gathered}$$
- How can we generalize it?
  - We need variables.
    $$\forall \alpha, \rho, \varphi, \varphi'.\texttt{exn}(\varphi) \xrightarrow{\varphi} \Box(\rho \rhd \alpha, \varphi')$$
    - Problem: set operations ($\cup, \setminus$), and subset order ($\subseteq$) for variables.
    - Two approaches
      - Bounded polymorphism
      - Row polymorphism ← our approache

Row Polymorphism

- Record type
  - $f = \lambda x.(x.a) : \{a : \texttt{int}\} \rightarrow \texttt{int}$
  - $f\ \{a : 1\} : \texttt{int}$ - O.K.
  - $f\ \{a : 1, b : \texttt{true}\}$: - Error!!
- Row polymorphism
  - $f = \lambda x.(x.a) : \{a : \texttt{int}; \rho\} \rightarrow \texttt{int}$
  - $f\ \{a : 1\} : \texttt{int}$ - O.K.
  - $f\ \{a : 1, b : \texttt{true}\} : \texttt{int}$ - O.K.
  - ρ represents all of the extra fields
- Set operations ($\cup, \setminus$) and subset order ($\subseteq$): Unification

Type and effect

$$
\begin{array}{rcll}
A, B & \in & \mathit{Type} & ::= \quad \alpha \mid \mathtt{int} \mid \mathtt{exn}(E) \mid A \xrightarrow{E} B \mid \Box(\Gamma \rhd A, E) \\
\alpha, \beta & \in & \mathit{TyVar} & \\
\Gamma & \in & \mathit{TyEnv} & ::= \quad \varnothing \mid \rho \mid x : F; \Gamma \\
\rho & \in & \mathit{TyEnvVar} & \\
F & \in & \mathit{Field} & ::= \quad \theta \mid A \\
\theta & \in & \mathit{FieldVar} & \\
E & \in & \mathit{Effect} & ::= \quad \varphi \mid c : \pi; E \\
\varphi & \in & \mathit{EffectVar} & \\
\pi & \in & \mathit{Presence} & ::= \quad \mathtt{Pre} \mid \delta \\
\delta & \in & \mathit{PresenceVar} &
\end{array}
$$

Type and effect

$$
\begin{array}{lcll}
\tau & \in & \mathit{TyScheme} & ::= \quad \forall \xi.\tau \mid A \\
\xi & \in & \mathit{Var} & ::= \quad \alpha \mid \rho \mid \theta \mid \varphi \mid \delta \\
\mu & \in & \mathit{FieldScheme} & ::= \quad \theta \mid \tau \\
\Delta & \in & \mathit{TySchemeEnv} & ::= \quad \rho \mid x : \mu; \Delta
\end{array}
$$

Typing judgment

$$\Delta_0 \cdots \Delta_n \vdash e : A, E_n \cdots E_0$$

Typing Rules

$$\frac{\Delta_n(x) \succ A}{\Delta_0 \cdots \Delta_n \vdash x : A, E_n \cdots E_0} \quad \texttt{(TVAR)}$$

$$\frac{\Delta_0 \cdots \Delta_n + x : A \vdash e : B, EE_{n-1} \cdots E_0}{\Delta_0 \cdots \Delta_n \vdash \lambda x.e : A \xrightarrow{E} B, E_n \cdots E_0} \quad \texttt{(TABS)}$$

$$\frac{\begin{array}{c}\Delta_0 \cdots \Delta_n \vdash e_1 : A \xrightarrow{E} B, EE_{n-1} \cdots E_0 \\ \Delta_0 \cdots \Delta_n \vdash e_2 : A, EE_{n-1} \cdots E_0\end{array}}{\Delta_0 \cdots \Delta_n \vdash e_1\ e_2 : B, EE_{n-1} \cdots E_0} \quad \texttt{(TAPP)}$$

$$\frac{\begin{array}{c}\Delta_0 \cdots \Delta_n \vdash e_1 : A, E_n \cdots E_0 \\ \Delta_0 \cdots \Delta_n + x : \mathit{GEN}_A(\Delta_0 \cdots \Delta_n, E_n \cdots E_0) \vdash e_2 : B, E_n \cdots E_0\end{array}}{\Delta_0 \cdots \Delta_n \vdash \texttt{let}\ (x\ e_1)\ e_2 : B, E_n \cdots E_0} \quad \texttt{(TLET)}$$

$\mathit{GEN}_A(\Delta_0 \cdots \Delta_n, E_n \cdots E_0) = \forall \xi_1 \cdots \xi_n.A$ such that
$\{\xi_1 \cdots \xi_n\} = FV(A) \setminus (FV(\Delta_0 \cdots \Delta_n) \cup FV(E_n \cdots E_0))$

Typing Rules

$$\frac{\Delta_0 \cdots \Delta_n \Delta \vdash e : A, E E_n \cdots E_0}{\Delta_0 \cdots \Delta_n \vdash \texttt{box}\, e : \Box(\Gamma \rhd A, E), E_n \cdots E_0} \quad \texttt{(TBOX)}$$

$$\frac{\Delta_0 \cdots \Delta_n \vdash e : \Box(\Gamma_{n+k} \rhd A, E_{n+k}), E_n \cdots E_0 \quad \Delta_{n+k} \succ \Gamma_{n+k}}{\Delta_0 \cdots \Delta_n \cdots \Delta_{n+k} \vdash \texttt{unbox}_k\, e : A, E_{n+k} \cdots E_n \cdots E_0} \quad \texttt{(TUNBOX)}$$

$$\frac{\Delta_0 \cdots \Delta_n \vdash e : \Box(\varnothing \rhd A, E), E E_{n-1} \cdots E_0}{\Delta_0 \cdots \Delta_n \vdash \texttt{eval}\, e : A, E E_{n-1} \cdots E_0} \quad \texttt{(TEVAL)}$$

- Rank-1 polymorphism:

  Arguments and results of a function can not be polymorphic.

Typing Rules

$$\frac{}{\Delta_0 \cdots \Delta_n \vdash c : \mathtt{exn}(c : \mathtt{Pre}; E), E_n \cdots E_0} \quad (\mathtt{TEXN})$$

$$\frac{\Delta_0 \cdots \Delta_n \vdash e : \mathtt{exn}(E), EE_{n-1} \cdots E_0}{\Delta_0 \cdots \Delta_n \vdash \mathtt{raise}\, e : A, EE_{n-1} \cdots E_0} \quad (\mathtt{TRAISE})$$

$$\frac{\begin{array}{c}\Delta_0 \cdots \Delta_n \vdash e_1 : A, (c : \mathtt{Pre}; E)E_{n-1} \cdots E_0 \\ \Delta_0 \cdots \Delta_n \vdash e_2 : A, (c : \pi; E)E'_{n-1} \cdots E'_0\end{array}}{\Delta_0 \cdots \Delta_n \vdash \mathtt{handle}\, e_1\ c\ e_2 : A, (c : \pi; E)E_{n-1} \cdots E_0} \quad (\mathtt{THANDLE})$$

Example

$$\rho \vdash \lambda x.\mathtt{box}\,(\mathtt{unbox}_1\,(\mathtt{raise}\,x)) : \alpha, \varphi \quad \{\alpha = \alpha_1 \xrightarrow{\varphi_1} \alpha_2\}$$
$$\rho_1 \vdash \mathtt{box}\,(\mathtt{unbox}_1\,(\mathtt{raise}\,x)) : \alpha_2, \varphi_1 \quad \{\rho_1 = (x : \alpha_1; \rho_2), \rho_2 = \rho\}$$
$$\rho_1\rho_3 \vdash \mathtt{unbox}_1\,(\mathtt{raise}\,x) : \alpha_3, \varphi_2\varphi_1 \quad \{\alpha_2 = \Box(\rho_3 \rhd \alpha_3, \varphi_2)\}$$
$$\rho_1 \vdash \mathtt{raise}\,x : \Box(\rho_3 \rhd \alpha_4, \varphi_2), \varphi_1$$
$$\rho_1 \vdash x : \mathtt{exn}(\varphi_1), \varphi_1 \quad \{(x : \alpha_1; \rho_2) = (x : \mathtt{exn}(\varphi_1); \rho_4)\}$$

$$\{\alpha_1 = \mathtt{exn}(\varphi_1)\}$$
$$\{\alpha = \mathtt{exn}(\varphi_1) \xrightarrow{\varphi_1} \Box(\rho_3 \rhd \alpha_3, \varphi_2)\}$$

$$\rho \vdash \lambda x.\mathtt{box}\,(\mathtt{unbox}_1\,(\mathtt{raise}\,x)) : \mathtt{exn}(\varphi_1) \xrightarrow{\varphi_1} \Box(\rho_3 \rhd \alpha_3, \varphi_2), \varphi$$

Example

$$\rho \vdash \texttt{let}\ (f\ \lambda x.\texttt{box}\ (\texttt{unbox}_1\ (\texttt{raise}\ x)))\ f\ c : \alpha, \varphi$$

$$\rho \vdash \lambda x.\texttt{box}\ (\texttt{unbox}_1\ (\texttt{raise}\ x)) : \alpha_1, \varphi$$
$$\{\alpha_1 = \texttt{exn}(\varphi_1) \xrightarrow{\varphi_1} \Box(\rho_1 \rhd \alpha_2, \varphi_2)\}$$

$$\rho_2 \vdash f\ c : \alpha, \varphi$$
$$\{\rho_2 = f : \forall \alpha_2 \rho_1 \varphi_1 \varphi_2.\texttt{exn}(\varphi_1) \xrightarrow{\varphi_1} \Box(\rho_1 \rhd \alpha_2, \varphi_2); \rho_3, \rho_3 = \rho\}$$

$$\rho_2 \vdash f : \alpha_3 \xrightarrow{\varphi} \alpha, \varphi$$
$$\{\varphi = \varphi_1, \alpha_3 = \texttt{exn}(\varphi_1), \alpha = \Box(\rho_1 \rhd \alpha_2, \varphi_2)\}$$

$$\rho_2 \vdash c : \alpha_3, \varphi$$
$$\{\alpha_3 = \texttt{exn}(c : \texttt{Pre}; \varphi_3), \varphi_1 = (c : \texttt{Pre}; \varphi_3)\}$$

$\{\alpha = \Box(\rho_1 \rhd \alpha_2, \varphi_2)\}$
$\{\varphi = (c : \texttt{Pre}; \varphi_3)\}$ May-uncaught exception c
$\rho \vdash \texttt{let}\ (f\ \lambda x.\texttt{box}\ (\texttt{unbox}_1\ (\texttt{raise}\ x)))\ f\ c : \Box(\rho_1 \rhd \alpha_2, \varphi_2), (c : \texttt{Pre}; \varphi_3)$

Example

$$\rho \vdash \begin{array}{l} \texttt{let} \\ \quad (f\ \lambda x.\texttt{box}\,(\texttt{unbox}_1\,(\texttt{raise}\,x))) \\ \texttt{handle}\,(f\ c)\ c\,(\texttt{box}\,1) \end{array} : \alpha, \varphi$$

$$\rho \vdash \lambda x.\texttt{box}\,(\texttt{unbox}_1\,(\texttt{raise}\,x)) : \alpha_1, \varphi$$
$$\{\alpha_1 = \texttt{exn}(\varphi_1) \xrightarrow{\varphi_1} \Box(\rho_1 \rhd \alpha_2, \varphi_2)\}$$

$$\rho_2 \vdash \texttt{handle}\,(f\ c)\ c\,(\texttt{box}\,1) : \alpha, \varphi$$
$$\{\rho_2 = f : \forall \alpha_2 \rho_1 \varphi_1 \varphi_2.\texttt{exn}(\varphi_1) \xrightarrow{\varphi_1} \Box(\rho_1 \rhd \alpha_2, \varphi_2); \rho_3, \rho_3 = \rho\}$$

$$\rho_2 \vdash f\ c : \alpha, \varphi$$
$$\{\alpha = \Box(\rho_1 \rhd \alpha_2, \varphi_2), (c : \texttt{Pre}; \varphi_4) = (c : \texttt{Pre}; \varphi_3)\}$$

$$\rho_2 \vdash \texttt{box}\,1 : \alpha, (c : \delta; \varphi_4)$$
$$\{\varphi = (c : \delta; \varphi_4), \alpha = \Box(\rho_3 \rhd \alpha_4, \varphi_5)\}$$

$$\rho_2\rho_3 \vdash 1 : \alpha_4, \varphi_5(c : \delta; \varphi_4)$$
$$\{\alpha_4 = \texttt{int}\}$$

$$\{\alpha = \Box(\rho_3 \rhd \texttt{int}, \varphi_5)\}$$
$$\{\varphi = (c : \delta; \varphi_3)\}$$

No uncaught exception!!

Conclusion

Conclusion

A type system for λ-calculus + Lisp's quasi-quote + exception

- exception-raise and -handle can appear at any stage
- exceptions (raised during code composition) can escape stages
- our effect type system safely supports such features
  - empty effect implies no uncaught exceptions