Exceptions are invaluable for structured error handling in high-level languages, but they are at odds with linear types. More generally, control effects may delete or duplicate portions of the stack, which, if we are not careful, can invalidate all substructural usage guarantees for values on the stack.

A Theory of
Substructural Types & Control

Jesse A. Tov   Riccardo Pucella

OOPSLA
October 26, 2011

Control Operators

exceptions, call/cc, shift and reset, coroutines, ...

Substructural Types

linear types, affine types, typestate, session types, ...

Substructural Types

Linear

Relevant

Affine

Unlimited

Substructural Types

L = 1

Relevant

Affine

Unlimited

Substructural Types

$$\begin{tikzcd}[column sep=small] & \mathrm{L} = 1 \arrow[dl, dash] \arrow[dr, dash] & \\ \mathrm{R} \geq 1 \arrow[dr, dash] & & \text{Affine} \arrow[dl, dash] \\ & \text{Unlimited} & \end{tikzcd}$$

Substructural Types

$$\begin{tikzcd}
 & \mathsf{L} = 1 \arrow[dl, dash] \arrow[dr, dash] & \\
\mathsf{R} \geq 1 \arrow[dr, dash] & & \mathsf{A} \leq 1 \arrow[dl, dash] \\
 & \text{Unlimited} &
\end{tikzcd}$$

Substructural Types

$$
\begin{tikzcd}
 & L = 1 \arrow[dl, dash] \arrow[dr, dash] & \\
R \geq 1 \arrow[dr, dash] & & A \leq 1 \arrow[dl, dash] \\
 & U &
\end{tikzcd}
$$

Substructural Types

type file : A

val *open* : string → file

val *read* : file → file × char

val *write* : file × char → file

val *close* : file → unit

Substructural Types

type file : **L**

val *open* : string $\to$ file

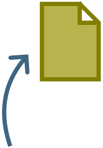
val *read* : file $\to$ file $\times$ char

val *write* : file $\times$ char $\to$ file

val *close* : file $\to$ unit

```
let confFile         = open confFileName in
let (conf, confFile) = parseConfFile confFile in
let logFile          = open conf.logFileName in
  close confFile;
  logFile
```

```
let confFile          = #⟨file:conf⟩ in
let (conf, confFile) = parseConfFile confFile in
let logFile           = open conf.logFileName in
  close confFile;
  logFile
```



```
let (conf, confFile) = parseConfFile #⟨file:conf⟩ in
let logFile          = open conf.logFileName in
   close confFile;
   logFile
```

```
let (conf, confFile) = ({ . . . }, #⟨file:conf⟩) in
let logFile          = open conf.logFileName in
   close confFile;
   logFile
```

let *logFile* = *open* { . . . }.*logFileName* **in**
  close #⟨file:conf⟩;
  logFile

let *logFile* = *open* “`/var/log/...`” **in**
    close #⟨file:conf⟩;
    logFile

raise *IOError*

	exceptions
affine types	☺
linear types	☹

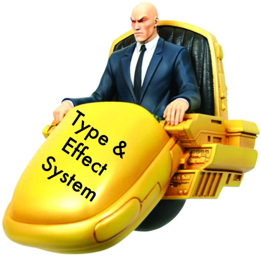
	exceptions	shift/reset
affine types	🙂	😐
linear types	☹	😐

(Danvy & Filinski 1989)

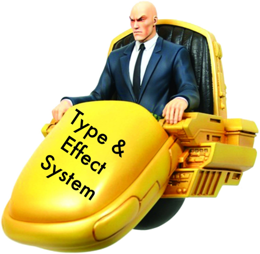
	exceptions	shift/reset
affine types	🙂	😐 → 🙂
linear types	☹️ → 🙂	😐 → 🙂




Type &
Effect
System



$$\Gamma \vdash e : \tau$$



$$\Gamma \vdash e : \tau \; ; \; c$$

λ^{URAL}

L

R Qualifiers A

U

(Ahmed et al. 2005)

λ^{URAL}

$$\begin{tikzcd} & \mathrm{L} \arrow[dl, dash] \arrow[dr, dash] & \\ \mathrm{R} \arrow[dr, dash] & \text{Qualifiers} & \mathrm{A} \arrow[dl, dash] \\ & \mathrm{U} & \end{tikzcd}$$

$$\vdash \tau \preceq Q$$

(Ahmed et al. 2005)

λ^{URAL}

L

R Qualifiers A

U

$$\vdash \tau \preceq Q$$

$$\vdash \Gamma \preceq Q$$

(Ahmed et al. 2005)

$\lambda^{\mathsf{URAL}}(\mathcal{C})$

$$\mathcal{C} = (\mathsf{C}, \perp, \oslash, \succeq)$$

$$\lambda^{\mathrm{URAL}}(\mathcal{C})$$

$$\mathcal{C} = (C, \perp, \oslash, \succeq)$$

effect names: $C \ni c$

$$\lambda^{\mathsf{URAL}}(\mathcal{C})$$

$$\mathcal{C} = (C, \bot, \oslash, \succeq)$$

effect names: $C \ni c$

pure effect: $\bot \in C$

$\lambda^{\mathsf{URAL}}(\mathcal{C})$

$$\mathcal{C} = (C, \bot, \otimes, \succeq)$$

effect names:	C	$\ni c$
pure effect:	$\bot$	$\in C$
sequencing:	$\otimes$	$: C \times C \rightharpoonup C$

$\lambda^{\mathsf{URAL}}(\mathcal{C})$

$$\mathcal{C} = (C, \bot, \otimes, \succeq)$$

$$\begin{array}{rll}
\text{effect names: } & C & \ni c \\
\text{pure effect: } & \bot & \in C \\
\text{sequencing: } & \otimes & : C \times C \rightharpoonup C \\
\text{qualifier bound: } & \succeq & \subseteq C \times Q
\end{array}$$

$\lambda^{\text{URAL}}(\mathcal{C})$

$$\mathcal{C} = (C, \perp, \otimes, \succeq)$$

	exceptions
effect names: C	$\mathcal{P}(\mathit{Exn})$
pure effect: $\perp$	$\varnothing$
sequencing: $\otimes$	$\cup$
qualifier bound: $\succeq$	$\vdash \{\phi\} \succeq A$

$\lambda^{\mathrm{URAL}}(\mathcal{C})$

$$\mathcal{C} = (\mathsf{C}, \bot, \oslash, \succeq)$$

	exceptions	shift/reset
effect names: C	$\mathcal{P}(\mathit{Exn})$	$\{\mathsf{U}, \mathsf{R}, \mathsf{A}, \mathsf{L}\}$
pure effect: $\bot$	$\varnothing$	L
sequencing: $\oslash$	$\cup$	$\sqcap$
qualifier bound: $\succeq$	$\vdash \{\phi\} \succeq \mathsf{A}$	$\vdash \mathsf{Q} \succeq \mathsf{Q}$

Application

$$\frac{}{\Gamma \quad \vdash e_1\ e_2}$$

Application

(check e_1) $\quad \Gamma \vdash e_1 : \tau' \multimap \tau$

(check e_2) $\quad \Gamma \vdash e_2 : \tau'$

$$\frac{\Gamma \vdash e_1 : \tau' \multimap \tau \qquad \Gamma \vdash e_2 : \tau'}{\Gamma \vdash e_1\, e_2 : \tau}$$

Context Splitting

$$
\begin{array}{ll}
(\text{check } e_1) & \Gamma_1 \vdash e_1 : \tau' \multimap \tau \\
(\text{check } e_2) & \Gamma_2 \vdash e_2 : \tau' \\
\hline
& \Gamma_1 \boxplus \Gamma_2 \vdash e_1\ e_2 : \tau
\end{array}
$$

Qualifier

$$
\begin{array}{ll}
(\text{check } e_1) & \Gamma_1 \vdash e_1 : {}^{Q_1}(\tau' \multimap \tau) \\
(\text{check } e_2) & \Gamma_2 \vdash e_2 : \tau' \\
\hline
& \Gamma_1 \boxplus \Gamma_2 \vdash e_1\, e_2 : \tau
\end{array}
$$

Control Effects

(check e_1) $\Gamma_1 \vdash e_1 : {}^{Q_1}(\tau' \xrightarrow{c} \tau) \; ; \; c_1$

(check e_2) $\Gamma_2 \vdash e_2 : \tau' \; ; \; c_2$

$$\frac{}{\Gamma_1 \boxplus \Gamma_2 \vdash e_1 \; e_2 : \tau}$$

Control Effects

(check e_1) $$\Gamma_1 \vdash e_1 : {}^{Q_1}(\tau' \overset{c}{\multimap} \tau) \; ; \; c_1$$

(check e_2) $$\Gamma_2 \vdash e_2 : \tau' \; ; \; c_2$$

(net effect) $$\vdash c_1 \ogreaterthan c_2 \ogreaterthan c : \mathrm{CTL}$$

$$\Gamma_1 \boxplus \Gamma_2 \vdash e_1 \; e_2 : \tau$$

Control Effects

(check e_1) $\Gamma_1 \vdash e_1 : {}^{Q_1}(\tau' \xrightarrow{c} \tau) \; ; \; c_1$

(check e_2) $\Gamma_2 \vdash e_2 : \tau' \; ; \; c_2$

(net effect) $\vdash c_1 \circledcirc c_2 \circledcirc c : \mathrm{CTL}$

$$\frac{}{\Gamma_1 \boxplus \Gamma_2 \vdash e_1\, e_2 : \tau \; ; \; c_1 \circledcirc c_2 \circledcirc c}$$

Effect of e_2

(check e_1) $\Gamma_1 \vdash e_1 : Q_1(\tau' \overset{c}{\multimap} \tau) \,;\, c_1$

(check e_2) $\Gamma_2 \vdash e_2 : \tau' \,;\, c_2$

(net effect) $\vdash c_1 \ogreaterthan c_2 \ogreaterthan c : \mathrm{CTL}$

$$\frac{}{\Gamma_1 \boxplus \Gamma_2 \vdash e_1\; e_2 : \tau \,;\, c_1 \ogreaterthan c_2 \ogreaterthan c}$$

Effect of e_2

$$
\begin{array}{rc}
\text{(check } e_1\text{)} & \Gamma_1 \vdash e_1 : {}^{Q_1}(\tau' \overset{c}{\multimap} \tau) \,;\, c_1 \\
\text{(check } e_2\text{)} & \Gamma_2 \vdash e_2 : \tau' \,;\, c_2 \\
\text{(}e_2\text{ effect ok)} & \vdash c_2 \succeq Q_1 \\
 & \\
\text{(net effect)} & \vdash c_1 \ogreaterthan c_2 \ogreaterthan c : \mathrm{CTL} \\
\hline
 & \Gamma_1 \boxplus \Gamma_2 \vdash e_1\; e_2 : \tau \,;\, c_1 \ogreaterthan c_2 \ogreaterthan c
\end{array}
$$

Effect of e_1

(check e_1)	$\Gamma_1 \vdash e_1 : {}^{Q_1}(\tau' \xrightarrow{c} \tau)\ ;\ c_1$
(check e_2)	$\Gamma_2 \vdash e_2 : \tau'\ ;\ c_2$
(e_2 effect ok)	$\vdash c_2 \succeq Q_1$
(net effect)	$\vdash c_1 \oslash c_2 \oslash c : \mathrm{CTL}$

$$\Gamma_1 \boxplus \Gamma_2 \vdash e_1\ e_2 : \tau\ ;\ c_1 \oslash c_2 \oslash c$$

Effect of e_1

$$
\begin{array}{rc}
\text{(check } e_1\text{)} & \Gamma_1 \vdash e_1 : {}^{Q_1}(\tau' \stackrel{c}{\multimap} \tau) \;;\; c_1 \\
\text{(check } e_2\text{)} & \Gamma_2 \vdash e_2 : \tau' \;;\; c_2 \\
\text{(}e_2\text{ effect ok)} & \vdash c_2 \succeq Q_1 \\
\text{(}e_2\text{ resources)} & \vdash \Gamma_2 \preceq Q_2 \\
\text{(}e_1\text{ effect ok)} & \vdash c_1 \succeq Q_2 \\
\text{(net effect)} & \vdash c_1 \ogreaterthan c_2 \ogreaterthan c : \mathit{CTL} \\
\hline
 & \Gamma_1 \boxplus \Gamma_2 \vdash e_1\, e_2 : \tau \;;\; c_1 \ogreaterthan c_2 \ogreaterthan c
\end{array}
$$

Application

$$
\frac{
\begin{array}{rc}
\text{(check } e_1\text{)} & \Gamma_1 \vdash e_1 : {}^{Q_1}(\tau' \overset{c}{\multimap} \tau) \; ; \; c_1 \\
\text{(check } e_2\text{)} & \Gamma_2 \vdash e_2 : \tau' \; ; \; c_2 \\
\text{(}e_2\text{ effect ok)} & \vdash c_2 \succeq Q_1 \\
\text{(}e_2\text{ resources)} & \vdash \Gamma_2 \preceq Q_2 \\
\text{(}e_1\text{ effect ok)} & \vdash c_1 \succeq Q_2 \\
\text{(net effect)} & \vdash c_1 \ogreaterthan c_2 \ogreaterthan c : \mathrm{CTL}
\end{array}
}{
\Gamma_1 \boxplus \Gamma_2 \vdash e_1\ e_2 : \tau \; ; \; c_1 \ogreaterthan c_2 \ogreaterthan c
}
$$

Does It Work?

```
let confFile           = open confFileName in
let (conf, confFile) = parseConfFile confFile in
let logFile            = open conf.logFileName in
  close confFile;
  logFile
```

Does It Work?

```
let confFile          = open confFileName in
let (conf, confFile)  = parseConfFile confFile in
close confFile;
let logFile           = open conf.logFileName in
  logFile
```

Does It Work?

Theorem (Type safety).
If $\bullet \vdash e : \tau\ ;\ \bot$ then *eval*(e) $\neq$ Wrong.

Proof (Parametrized by $\mathcal{C}$).
Transform e to continuation-passing style . . .

Does It Work?

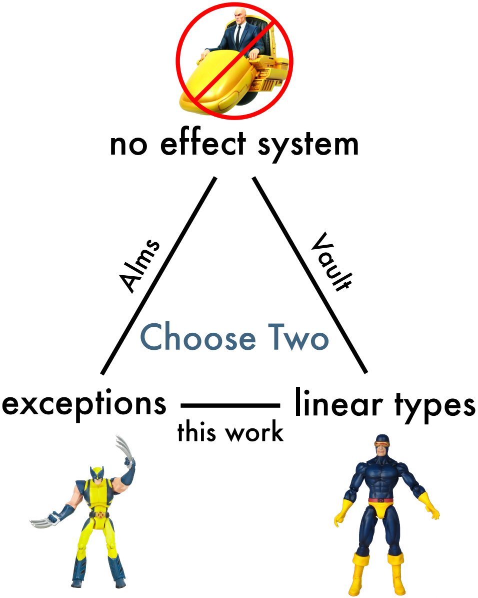
Theorem (Type safety).

If $\bullet \vdash \mathsf{e} : \tau \,;\, \bot$ then *eval*(e) $\neq$ Wrong.

Proof (Parametrized by $\mathcal{C}$).

Transform e to continuation-passing style . . .

Three instances for $\mathcal{C}$: exceptions, shift/reset, and shift/reset with answer-type modification


no effect system
Alms
Vault
Choose Two
exceptions
this work
linear types

The Take-Away

Designing a substructural type system?
Considering adding control effects?

Read our paper

http://www.ccs.neu.edu/∼tov/pubs/