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 \hspace{1cm} Unlimited

Affine
Substructural Types

\[ L = 1 \]

\[ R \geq 1 \]

Unlimited

Affine
Substructural Types

\[ R \geq 1 \quad L = 1 \quad A \leq 1 \]

Unlimited
Substructural Types

\[
\begin{align*}
L &= 1 \\
R &\geq 1 \\
\quad U \\
A &\leq 1
\end{align*}
\]
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 → file
val read : file → file × char
val write : file × char → file
val close : file → 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
let confFile = ⟨file:.conf⟩ in let (conf, confFile) = ({...}, ⟨file:.conf⟩) in raise IOError
exceptions

affine types

linear types

(Danvy & Filinski 1989)
exceptions shift/reset

affine types

linear types

(Danvy & Filinski 1989)
exceptions shift/reset

affine types

linear types

(exceptions shift/reset)
The Type & Effect System
(Ahmed et al. 2005)
(Ahmed et al. 2005)
(Ahmed et al. 2005)
$\lambda_{URAL}(C)$

$C = (C, \bot, \emptyset, \succeq)$
\[ \lambda^{URAL}(\mathcal{C}) \]

\[ \mathcal{C} = (\mathcal{C}, \bot, \emptyset, \succeq) \]

effect names: \( \mathcal{C} \ni c \)
\[ \lambda^{\text{URAL}}(\mathcal{C}) \]

\[ \mathcal{C} = (\mathbf{C}, \bot, \emptyset, \succeq) \]

effect names: \( \mathbf{C} \ni c \)

pure effect: \( \bot \in \mathbf{C} \)
\[ \lambda_{\text{URAL}}(\mathcal{C}) \]

\[ \mathcal{C} = (\mathbb{C}, \perp, \emptyset, \succeq) \]

effect names: \( \mathbb{C} \ni c \)

pure effect: \( \perp \in \mathbb{C} \)

sequencing: \( \emptyset : \mathbb{C} \times \mathbb{C} \rightarrow \mathbb{C} \)
\[ \lambda^{URAL}(C) \]

\[ C = (C, \perp, \otimes, \succeq) \]

- effect names: \( C \ni c \)
- pure effect: \( \perp \in C \)
- sequencing: \( \otimes : C \times C \rightarrow C \)
- qualifier bound: \( \succeq \subseteq C \times Q \)
\[\lambda^\text{URAL}(C)\]

\[C = (C, \bot, \emptyset, \succeq)\]

exceptions

- effect names: \(C\)
- pure effect: \(\bot\)
- sequencing: \(\emptyset\)
- qualifier bound: \(\succeq\)

\[\mathcal{P}(\mathbb{E}xn)\]
\[\emptyset\]
\[\cup\]
\[\vdash \{\phi\} \succeq A\]
\[ \lambda^{URAL}(C) \]

\[ C = (C, \perp, \emptyset, \succeq) \]

- **effect names:** \( C \)
- **pure effect:** \( \perp \)
- **sequencing:** \( \emptyset \)
- **qualifier bound:** \( \succeq \)

**exceptions**

\[ P(Exn) \]

\[ \emptyset \]

\[ \cup \]

\[ \vdash \{ \phi \} \succeq A \]

**shift/reset**

\[ \{ U, R, A, L \} \]

\[ L \]

\[ \prod \]

\[ \vdash Q \succeq Q \]
Application

\[ \Gamma \vdash e_1 e_2 \]
Application

\[(\text{check } e_1)\]
\[(\text{check } e_2)\]

\[\Gamma \vdash e_1 : \tau' \rightarrow \tau\]
\[\Gamma \vdash e_2 : \tau'\]

\[\Gamma \vdash e_1 \ e_2 : \tau\]
Context Splitting

(\text{check } e_1)
\Gamma_1 \vdash e_1 : \tau' \rightarrow \tau

(\text{check } e_2)
\Gamma_2 \vdash e_2 : \tau'

\Gamma_1 \boxplus \Gamma_2 \vdash e_1 e_2 : \tau
(check e₁)

$\Gamma_1 \vdash e_1 : Q_1(\tau' \rightarrow \tau)$

(put e₂)

$\Gamma_2 \vdash e_2 : \tau'$

$\Gamma_1 \boxplus \Gamma_2 \vdash e_1 \cdot e_2 : \tau$
Control Effects

(check \( e_1 \)) \qquad \Gamma_1 \vdash e_1 : Q_1 (\tau' \circ c) ; c_1

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

\hline

\quad \Gamma_1 \boxdot \Gamma_2 \vdash e_1 e_2 : \tau
Control Effects

(check e₁)
\[ \Gamma_1 \vdash e_1 : Q_1 (T' \xrightarrow{c} T) ; c_1 \]

(check e₂)
\[ \Gamma_2 \vdash e_2 : T' ; c_2 \]

(net effect)
\[ \vdash c_1 \bigoplus c_2 \bigoplus c : CTL \]

\[ \Gamma_1 \boxplus \Gamma_2 \vdash e_1 e_2 : T \]
Control Effects

(\text{check } e_1) \quad \Gamma_1 \vdash e_1 : Q_1(T' \xrightarrow{c} T) ; c_1

(\text{check } e_2) \quad \Gamma_2 \vdash e_2 : T' ; c_2

(\text{net effect}) \quad \vdash c_1 \otimes c_2 \otimes c : CTL

\Gamma_1 \boxdot \Gamma_2 \vdash e_1 e_2 : T ; c_1 \otimes c_2 \otimes c
Effect of $e_2$

(check $e_1$)  

\[
\Gamma_1 \vdash e_1 : Q_1 (\tau' \overset{c}{\rightarrow} \tau) ; c_1
\]

(check $e_2$)  

\[
\Gamma_2 \vdash e_2 : \tau' ; c_2
\]

(net effect)  

\[
\vdash c_1 \otimes c_2 \otimes c : CTL
\]

\[
\Gamma_1 \boxdot \Gamma_2 \vdash e_1 e_2 : \tau ; c_1 \otimes c_2 \otimes c
\]
Effect of $e_2$

(check $e_1$)

$$\Gamma_1 \vdash e_1 : Q_1(\tau' \circ c \circ \tau) ; c_1$$

(check $e_2$)

$$\Gamma_2 \vdash e_2 : \tau' ; c_2$$

(e2 effect ok)

$$\vdash c_2 \supseteq Q_1$$

(net effect)

$$\vdash c_1 \otimes c_2 \otimes c : CTL$$

$$\Gamma_1 \boxplus \Gamma_2 \vdash e_1 e_2 : \tau ; c_1 \otimes c_2 \otimes c$$
Effect of $e_1$

(\textit{check }e_1) \quad \Gamma_1 \vdash e_1 : Q_1(\mathcal{T}' \xrightarrow{c} \mathcal{T}) \ ; c_1$

(\textit{check }e_2) \quad \Gamma_2 \vdash e_2 : \mathcal{T}' \ ; c_2

(e_2 \textit{ effect ok}) \quad \vdash c_2 \preceq Q_1$

(\textit{net effect}) \quad \vdash c_1 \otimes c_2 \otimes c : \text{CTL}$

\[ \Gamma_1 \boxdot \Gamma_2 \vdash e_1 e_2 : \mathcal{T} \ ; c_1 \otimes c_2 \otimes c \]
Effect of $e_1$

\[
\begin{align*}
\Gamma_1 &\vdash e_1 : Q_1(\tau' \circ \tau) ; c_1 \\
\Gamma_2 &\vdash e_2 : \tau' ; c_2 \\
&\vdash c_2 \preceq Q_1 \\
&\vdash \Gamma_2 \preceq Q_2 \\
&\vdash c_1 \preceq Q_2 \\
&\vdash c_1 \otimes c_2 \otimes c : CTL
\end{align*}
\]
Application

\(\Gamma_1 \vdash e_1 : Q_1(\tau' \circ \tau) ; c_1\)
\(\Gamma_2 \vdash e_2 : \tau' ; c_2\)

\(\vdash c_2 \succeq Q_1\)
\(\vdash \Gamma_2 \succeq Q_2\)
\(\vdash c_1 \succeq Q_2\)

\(\vdash c_1 \otimes c_2 \otimes \cdot : \text{CTL}\)

\(\Gamma_1 \boxplus \Gamma_2 \vdash e_1 e_2 : \tau ; c_1 \otimes c_2 \otimes \cdot\)
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?

```plaintext
let confFile = open confFileName in
let (conf, confFile) = parseConfFile confFile in
close confFile;
let logFile = open conf.logFileName in
logFile
```
Theorem (Type safety).
If $\cdot \vdash e : \tau ; \bot$ then $\text{eval}(e) \neq \text{Wrong}$.

Proof (Parametrized by $C$).
Transform $e$ to continuation-passing style . . .
Theorem (Type safety).
If $\bullet \vdash e : \tau ; \bot$ then $\text{eval}(e) \neq \text{Wrong}$.

Proof (Parametrized by $C$).
Transform $e$ to continuation-passing style . . .

Three instances for $C$: exceptions, shift/reset, and shift/reset with answer-type modification
Choose Two

exceptions  linear types

this work

no effect system

Alms

Vault
The Take-Away

Designing a substructural type system? Considering adding control effects?

Read our paper

http://www.ccs.neu.edu/~tov/pubs/