A logical interpretation of Java-style exceptions

Jeff Vaughan

Department of Computer Science
Harvard University

CL&C, August 22, 2010

Pop Quiz!

Ignore divergence and mutable state. What is the logical content of the following program?

```
D m(C arg) throws E F { ... }
```

Pop Quiz!

Ignore divergence and mutable state. What is the logical content of the following program?

```
D m(C arg) throws E F { ... }
```

Answer: $C \supset D \vee E \vee F$

There are many logic-based readings of exceptions.

- [Nakano '92, '94] Introduced exceptions in an extension of system LJ'.
- [Sato '97] A natural-deduction style logic with exceptions.
- [Kameyama '97] Exceptions in Gödel's T.
- [De Groote '95] Exceptions are named by lexically-scoped variables, with classical typing rules.
- [Ong & Steward '97] Exceptions names are covariables in μPFC.

⋮

Beautiful models, but far from practical languages.

- For example, Nakano '92:
  - Exception names are represented by lexically-scoped *tags*.
  - Many administrative tag abstractions and instantiations.
  - Latent effects must be manually suspended as part of function definitions.
- *Type-and-effect* analyses address these problems, but have received little attention from a logic perpsective.
  [Lucassen & Gifford '88, Talpin & Jouvelot '92]

Outline

This talk: Finding the logical-content of exceptions, from the perspective of type-and-effect analysis.

System EC: An exception calculus

EC models Java-style exceptions.

- Exceptions are first class values, and are identified by type name.
- Checked exception methodology requires that functions be annotated with a set of throwable exceptions.
- Subtyping lets one exception handler catch multiple related exceptions.
- Call-by-value semantics enable precise reasoning.

Focusing on exceptions: no classes, divergence, or state.

EC's expression language extends lambda calculus.

Definition (Expression Syntax)

e	$::=$	$x \mid e_1\ e_2 \mid \lambda x : \tau.\ e$	Lambda calculus
	$\mid$	**top**	Top/unit value
	$\mid$	$E\ e$	Exception expression
	$\mid$	**raise** e	Throw exception
	$\mid$	e_1 **handle** $E\ x \Rightarrow e_2$	Exception handler

EC's expression language extends lambda calculus.

Definition (Expression Syntax)

e	$::=$	$x \mid e_1\ e_2 \mid \lambda x : \tau.\ e$	Lambda calculus
	$\mid$	**top**	Top/unit value
	$\mid$	$E\ e$	Exception expression
	$\mid$	**raise** e	Throw exception
	$\mid$	e_1 **handle** $E\ x \Rightarrow e_2$	Exception handler

Example (Evaluation)

$$(\lambda x : \textbf{top}.\ \textbf{raise}\ \textit{Fail}\ \text{"}\textit{ohno}\text{!"})\ 1\ \textbf{handle}\ \textit{Any}\ x \Rightarrow 2$$

$$\rightarrow^*$$

$$2$$

Type-and-effects–style analysis tracks exceptions.

$$e \equiv \textbf{if } b \textbf{ then } 3 \textbf{ else } (\textbf{raise } \mathit{Fail} \text{ "}\mathit{oops}\text{"})$$

$$e : \quad \textbf{ret} : \mathit{int} \ , \ \textbf{exn } \mathit{Fail} : \mathit{string}$$

Type-and-effects–style analysis tracks exceptions.

$$e \equiv \textbf{if } b \textbf{ then } 3 \textbf{ else } (\textbf{raise } \mathit{Fail} \text{ "}\mathit{oops}\text{"})$$

$$e : \quad \textbf{ret} : \mathit{int} \ , \ \textbf{exn } \mathit{Fail} : \mathit{string}$$

Type of normal termination

Type-and-effects–style analysis tracks exceptions.

$$e \equiv \textbf{if } b \textbf{ then } 3 \textbf{ else } (\textbf{raise } \mathit{Fail} \text{ "}\mathit{oops}\text{"})$$

$$e : \quad \textbf{ret} : \mathit{int} \;,\; \textbf{exn } \mathit{Fail} : \mathit{string}$$

Type of normal termination

List of possible exceptions

Type-and-effects–style analysis tracks exceptions.

$$e \equiv \textbf{if } b \textbf{ then } 3 \textbf{ else } (\textbf{raise } \textit{Fail } \text{"}\textit{oops}\text{"})$$

$$e : \quad \textbf{ret} : \textit{int} \;,\; \textbf{exn } \textit{Fail} : \textit{string}$$

$$\lambda b : \textit{bool}.\, e : \quad \textbf{ret} \;: (\textit{bool} \rightarrow (\textbf{ret} : \textit{int},\; \textbf{exn } \textit{Fail} : \textit{string}\;))$$

Type-and-effects–style analysis tracks exceptions.

$$e \equiv \textbf{if}\ b\ \textbf{then}\ 3\ \textbf{else}\ (\textbf{raise}\ \mathit{Fail}\ \text{"}\mathit{oops}\text{"})$$

$$e:\quad \textbf{ret}: \mathit{int}\ ,\ \textbf{exn}\ \mathit{Fail}: \mathit{string}$$

$$\lambda b: \mathit{bool}.\, e:\quad \textbf{ret}\ : (\mathit{bool} \to (\textbf{ret}: \mathit{int},\ \textbf{exn}\ \mathit{Fail}: \mathit{string}\ ))$$

Values always return

Type-and-effects–style analysis tracks exceptions.

$$e \equiv \textbf{if } b \textbf{ then } 3 \textbf{ else } (\textbf{raise } \mathit{Fail}\ "\mathit{oops}")$$

$$e : \quad \textbf{ret} : \mathit{int}\ ,\ \textbf{exn } \mathit{Fail} : \mathit{string}$$

$$\lambda b : \mathit{bool}.\, e : \quad \textbf{ret}\ : (\mathit{bool} \rightarrow (\textbf{ret} : \mathit{int},\ \textbf{exn } \mathit{Fail} : \mathit{string}\ ))$$

Values always return

Function body exceptions are captured as *latent effects*

Exception typing uses signatures and result contexts.

Form of typing judgment

$$e : \Sigma ; \Gamma \vdash \Delta$$

Definition

Σ	$::=$	$\cdot$	Empty Signature
	$\mid$	$\Sigma, E \sim \tau$	Exception declaration
	$\mid$	$\Sigma, E_1 <: E_2$	Subtype declaration
Δ	$::=$	$\cdot \mid \Delta, \textbf{ret} : \tau \mid \Delta, \textbf{exn} : \tau$	Result Context
τ	$::=$	$A \mid \top$	Base types and top
	$\mid$	$\tau \to \Delta$	Arrow
	$\mid$	$\textbf{Exn}\ E$	Exception type

App & abs rules are specialized for latent effects

$$\frac{e : \Sigma; \Gamma, x : \tau \vdash \Delta}{\lambda x : \tau.\ e : \Sigma; \Gamma \vdash \mathbf{ret} : \tau \to \Delta}\ \textsc{T-Abs}$$

$$\frac{\begin{array}{c} e_1 : \Sigma; \Gamma \vdash \mathbf{ret} : \tau_2 \to (\mathbf{ret} : \tau, \Delta), \Delta \\ e_2 : \Sigma; \Gamma \vdash \mathbf{ret} : \tau_2, \Delta \end{array}}{e_1\ e_2 : \Sigma; \Gamma \vdash \mathbf{ret} : \tau, \Delta}\ \textsc{T-App}$$

* Computational effects are suspended at abstractions and restored at applications.

Subtyping is a useful programming language feature.

$$\Sigma \equiv \mathit{Disk} <: \mathit{IO}, \mathit{Sound} <: \mathit{IO}, \mathit{IO} <: \mathit{Any}, \ldots$$

$$e : \Sigma; \cdot \vdash \textbf{ret} : \mathit{int}, \textbf{exn}\ \mathit{Disk} : \mathit{string}, \textbf{exn}\ \mathit{Sound} : \top$$

Example (Polymorphic exception handling)

$$e\ \textbf{handle}\ \mathit{IO}\ x \Rightarrow e'$$

catches all *Disk*, *Sound*, and *IO* exceptions.

Example (Subtyping allows conservative typings)

$$e : \Sigma; \cdot \vdash \textbf{ret} : \mathit{int}, \textbf{exn}\ \mathit{Any} : \top$$

Embedding of EC in classical logic

EC typing derivations give rise to LK proofs trees.

- Each EC type corresponds to an LK proposition.
  - Mostly standard interpretation
    [Curry, Feys & Craig ’58, Howard ’80]
  - Latent effects are represented by disjunction.
- EC subtyping translates to logical entailment.
- EC typing derivations translate to LK derivations.

LK is the canonical classical sequent calculus.

Definition (LK Propositions)

$$P, Q \quad ::= \quad A_{\mathrm{LK}} \mid \top \mid \bot \mid P \wedge Q \mid P \vee Q \mid P \supset Q \mid \neg P$$

Judgment form

$$P_1 \ldots P_n \mapsto Q_1 \ldots Q_m$$

"means" the conjunction of the Ps implies the disjunction of the Qs.

[Gentzen '35]

EC types translate to LK propositions.

Definition ($[\![\cdot]\!]^{\cdot}$)

$$
\begin{aligned}
[\![A]\!]^{\Sigma}_{\mathbf{typ}} &= A_{\mathrm{LK}} \\
[\![\top]\!]^{\Sigma}_{\mathbf{typ}} &= \top \\
[\![\mathbf{Exn}\ E]\!]^{\Sigma}_{\mathbf{typ}} &= [\![\tau]\!]^{\Sigma}_{\mathbf{typ}} \qquad \text{where } E \sim \tau \in \Sigma \\
[\![\tau \to \Delta]\!]^{\Sigma}_{\mathbf{typ}} &= [\![\tau]\!]^{\Sigma}_{\mathbf{typ}} \supset \bigovee [\![\Delta]\!]^{\Sigma}_{\mathbf{ctx}}
\end{aligned}
$$

$$
\begin{aligned}
[\![\cdot]\!]^{\Sigma}_{\mathbf{ctx}} = [\![\cdot]\!]^{\Sigma}_{\mathbf{env}} &= \cdot \\
[\![\Gamma, \_ : \tau]\!]^{\Sigma}_{\mathbf{env}} &= [\![\Gamma]\!]^{\Sigma}_{\mathbf{env}}, [\![\tau]\!]^{\Sigma}_{\mathbf{typ}} \\
[\![\Delta, \_ : \tau]\!]^{\Sigma}_{\mathbf{ctx}} &= [\![\Delta]\!]^{\Sigma}_{\mathbf{ctx}}, [\![\tau]\!]^{\Sigma}_{\mathbf{typ}}
\end{aligned}
$$

$$
\bigovee P_1, P_2, \ldots, P_n = P_1 \vee P_2 \vee \cdots \vee P_n
$$

Main result: typing and subtyping have logical content.

Lemma (Subtyping)

Suppose $\mathscr{D}$ a subtyping derivation in EC.

- *If* $\mathscr{D} :: \Sigma \vdash \Delta_1 <: \Delta_2$ *then* $\bigcirc\!\!\!\!\vee [\![\Delta_1]\!]^{\Sigma}_{\mathbf{ctx}} \mapsto [\![\Delta_2]\!]^{\Sigma}_{\mathbf{ctx}}$.
- *If* $\mathscr{D} :: \Sigma \vdash \tau_1 <: \tau_2$ *then* $[\![\tau_1]\!]^{\Sigma}_{\mathbf{typ}} \mapsto [\![\tau_2]\!]^{\Sigma}_{\mathbf{typ}}$.
- *If* $\mathscr{D} :: \Sigma \vdash \diamond$ *then* $[\![\tau_1]\!]^{\Sigma}_{\mathbf{typ}} \mapsto [\![\tau_2]\!]^{\Sigma}_{\mathbf{typ}}$ *where* $E_1 <: E_2$, $E_1 \sim \tau_1, E_2 \sim \tau_2 \in \Sigma$.

Theorem (Shallow embedding of EC in LK)

Suppose $e : \Sigma; \Gamma \vdash \Delta$. *Then* $[\![\Gamma]\!]^{\Sigma}_{\mathbf{env}} \mapsto [\![\Delta]\!]^{\Sigma}_{\mathbf{ctx}}$.

Future directions

Cam we move from an embedding to an isomorphism?

- EC is (likely) constructive, but LK is classical—(likely) no way to translate LK proofs to EC typing derivations.

- LJ' may be a good translation target.
  - LJ' is an intuitionistic variant of LK.
  - LJ' has multiple conclusions—these seem essential.
  - LJ' restricts the LK implication and negation rules:

$$\frac{P, \Phi \mapsto \Theta, Q}{\Phi \mapsto \Theta, P \supset Q}\ \text{LK-IMPR} \qquad \frac{P, \Phi \mapsto Q}{\Phi \mapsto P \supset Q}\ \text{LJ'-IMPR}$$

  - Implication restriction looks like a good fit with EC's handling of latent effects.

[Takeuti '75]

Can we logically interpret other effects systems?

- Possible nontermination ⇒ Local absence of logical content?
  Termination casts [Stump, Sjöberg, and Weirich '10]

- World effects ⇒ An alternative means to model context?
  Contextual modal type theory [Nanevski, Pfenning, and Pientka '08]

Conclusions

- This talk: EC typing derivations give rise to LK proofs.
- Generally: Effects systems can have a logical interpretation.
- Many interesting problems remain!

Conclusions

- This talk: EC typing derivations give rise to LK proofs.
- Generally: Effects systems can have a logical interpretation.
- Many interesting problems remain!

Thank you!