# 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 { ... }
```

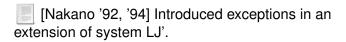
### Pop Quiz!

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

D m(C arg) throws E F 
$$\{ \ldots \}$$

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

### There are many logic-based readings of exceptions.



[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  $\mu$ PFC.

:

### 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.

- 1 System EC: An exception calculus
- 2 Embedding of EC in classical logic
- 3 Future directions

### 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)

$$e ::= x \mid e_1 \mid e_2 \mid \lambda x \colon \tau. \mid e$$
 Lambda calculus Top/unit value Exception expression Throw exception  $e_1 \mid a \mid b \mid e_2 \mid \lambda x \colon \tau. \mid e$  Lambda calculus Top/unit value Exception expression Exception handler

### EC's expression language extends lambda calculus.

#### Definition (Expression Syntax)

```
e ::= x \mid e_1 \mid e_2 \mid \lambda x \colon \tau. \mid e Lambda calculus Top/unit value Exception expression \mid \mathbf{raise} \mid e \mid e_1 \mid \mathbf{handle} \mid E \mid x \Rightarrow e_2 \mid e_2 \mid e_1 \mid e_2 \mid \lambda x \colon \tau. \mid e Lambda calculus Top/unit value Exception expression Exception handler
```

#### Example (Evaluation)

```
(\lambda x : 	extbf{top. raise } \textit{Fail "ohno!"}) \ 1 \ 	extbf{handle } \textit{Any } x \Rightarrow 2 \rightarrow^* 2
```

### Type-and-effects—style analysis tracks exceptions.

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

e: ret: int , exn Fail: string

### Type-and-effects-style analysis tracks exceptions.

```
e ≡ if b then 3 else (raise Fail "oops")

e: ret: int , exn Fail: string

Type of normal termination
```

### Type-and-effects-style analysis tracks exceptions.

 $e \equiv \text{if } b \text{ then } 3 \text{ else } (\text{raise } \textit{Fail "oops"})$   $e : \quad \text{ret} : \textit{int }, \text{ exn } \textit{Fail} : \textit{string}$   $Type \text{ of } \\ \text{normal} \\ \text{termination}$   $\text{List of } \\ \text{possible} \\ \text{exceptions}$ 

### Type-and-effects—style analysis tracks exceptions.

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

e: ret: int , exn Fail: string

 $\lambda b$ : bool. e: ret : (bool  $\rightarrow$  (ret: int, exn Fail: string))

### Type-and-effects-style analysis tracks exceptions.

```
e \equiv \text{if } b \text{ then } 3 \text{ else } (\text{raise } Fail "oops")
                       ret: int , exn Fail: string
\lambda b: bool. e: ret : (bool \rightarrow (ret: int, exn Fail: string))
           Values
          always
```

### Type-and-effects-style analysis tracks exceptions.

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

e: ret: int , exn Fail: string

 $\lambda b \colon bool.\ e \colon \operatorname{ret} : (bool \to (\operatorname{ret} \colon \operatorname{int}, \operatorname{exn} \operatorname{\it Fail} \colon \operatorname{\it string}\,))$ 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

$$\Sigma ::= \cdot$$
 $| \Sigma, E \sim \tau$ 
 $| \Sigma, E_1 <: E_2$ 

$$\Delta ::= \cdot | \Delta, \mathbf{ret} : \tau | \Delta, \mathbf{exn} : \tau$$
 Result Context

$$\begin{array}{ccc} \tau & ::= & A \mid \top \\ & \mid & \tau \rightarrow \Delta \\ & \mid & \mathbf{Exn} \ E \end{array}$$

**Empty Signature** Exception declaration Subtype declaration

Base types and top Arrow Exception type

### App & abs rules are specialized for latent effects

$$\frac{\textit{e} \colon \Sigma; \Gamma, \textit{x} \colon \tau \vdash \Delta}{\lambda \textit{x} \colon \tau. \; \textit{e} \colon \Sigma; \Gamma \vdash \textbf{ret} \colon \tau \to \Delta} \; \text{T-Abs}$$

$$\begin{aligned} & e_1 \colon \Sigma; \Gamma \vdash \textbf{ret} \colon \tau_2 \to (\textbf{ret} \colon \tau, \Delta), \Delta \\ & \frac{e_2 \colon \Sigma; \Gamma \vdash \textbf{ret} \colon \tau_2, \Delta}{e_1 \ e_2 \colon \Sigma; \Gamma \vdash \textbf{ret} \colon \tau, \Delta} \end{aligned} \text{ T-App}$$

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

### Subtyping is a useful programming language feature.

$$\Sigma \equiv \textit{Disk} <: \textit{IO}, \textit{Sound} <: \textit{IO}, \textit{IO} <: \textit{Any}, \dots$$

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

#### Example (Polymorphic exception handling)

*e* handle 
$$IO x \Rightarrow e'$$

catches all Disk, Sound, and IO exceptions.

#### Example (Subtyping allows conservative typings)

$$e: \Sigma; \cdot \vdash ret: int, exn Any: \top$$

### 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)

$$P,Q$$
 ::=  $A_{LK} \mid \top \mid \bot \mid P \land Q \mid P \lor Q \mid P \supset Q \mid \neg P$ 

Judgment form

$$P_1 \dots P_n \mapsto Q_1 \dots Q_m$$

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



### EC types translate to LK propositions.

### Definition ([[·]]·)

$$\begin{split} & [\![A]\!]_{\mathsf{typ}}^{\Sigma} &= A_{\mathsf{LK}} \\ & [\![\top]\!]_{\mathsf{typ}}^{\Sigma} &= \top \\ & [\![\mathsf{Exn}\,E]\!]_{\mathsf{typ}}^{\Sigma} &= [\![\tau]\!]_{\mathsf{typ}}^{\Sigma} \quad \text{where } E \sim \tau \in \Sigma \\ & [\![\tau \to \Delta]\!]_{\mathsf{typ}}^{\Sigma} &= [\![\tau]\!]_{\mathsf{typ}}^{\Sigma} \supset \varnothing [\![\Delta]\!]_{\mathsf{ctx}}^{\Sigma} \\ & [\![\cdot]\!]_{\mathsf{ctx}}^{\Sigma} &= [\![\cdot]\!]_{\mathsf{env}}^{\Sigma} &= \cdot \\ & [\![\Gamma, ...; \tau]\!]_{\mathsf{env}}^{\Sigma} &= [\![\Gamma]\!]_{\mathsf{env}}^{\Sigma}, [\![\tau]\!]_{\mathsf{typ}}^{\Sigma} \\ & [\![\Delta, ...; \tau]\!]_{\mathsf{ctx}}^{\Sigma} &= [\![\Delta]\!]_{\mathsf{ctx}}^{\Sigma}, [\![\tau]\!]_{\mathsf{typ}}^{\Sigma} \\ & \varnothing P_{1}, P_{2}, \dots, P_{n} &= P_{1} \lor P_{2} \lor \dots \lor P_{n} \end{split}$$

### 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  $\otimes [\![\Delta_1]\!]_{\mathbf{ctx}}^{\Sigma} \mapsto [\![\Delta_2]\!]_{\mathbf{ctx}}^{\Sigma}$ .
- If  $\mathscr{D}$  ::  $\Sigma \vdash \tau_1 <: \tau_2$  then  $[[\tau_1]]_{\mathsf{typ}}^{\Sigma} \mapsto [[\tau_2]]_{\mathsf{typ}}^{\Sigma}$ .
- If  $\mathscr{D}$  ::  $\Sigma \vdash \diamond$  then  $[[\tau_1]]_{\mathsf{typ}}^{\Sigma} \mapsto [[\tau_2]]_{\mathsf{typ}}^{\Sigma}$  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  $[\![\Gamma]\!]_{env}^{\Sigma} \mapsto [\![\Delta]\!]_{ctx}^{\Sigma}$ .

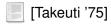
### **Future directions**

### 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:

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

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



### 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!