A Quantitative Understanding of Exceptions

Miguel Ramos [1] Riccardo Treglia [2] Delia Kesner [3]

TLLA 2023

[1]LIACC, DCC, Faculdade de Ciências, Universidade do Porto
[2]Università di Bologna
[3]IRIF, CNRS, Université Paris Cité & Institut Universitaire de France

Programming Languages

λ-calculus (Pure)

Real (Impure)

Programming Languages

λ-calculus (Pure)

- Simple structure

Real (Impure)

Programming Languages

λ-calculus (Pure)

- Simple structure

Real (Impure)

- Complicated structure

Programming Languages

λ-calculus (Pure)

- Simple structure
- No side-effects

Real (Impure)

- Complicated structure

Programming Languages

λ-calculus (Pure)

- Simple structure
- No side-effects

Real (Impure)

- Complicated structure
- Side-effects

Programming Languages

λ-calculus (Pure)

- Simple structure
- No side-effects
- Easy to reason about

Real (Impure)

- Complicated structure
- Side-effects

Programming Languages

λ-calculus (Pure)

- Simple structure
- No side-effects
- Easy to reason about

Real (Impure)

- Complicated structure
- Side-effects
- Hard to reason about

Programming Languages

λ-calculus (Pure)

- Simple structure
- No side-effects
- Easy to reason about
- Useless for programmers

Real (Impure)

- Complicated structure
- Side-effects
- Hard to reason about

Programming Languages

λ-calculus (Pure)

- Simple structure
- No side-effects
- Easy to reason about
- Useless for programmers

Real (Impure)

- Complicated structure
- Side-effects
- Hard to reason about
- Interact with the real world

Programming Languages

λ-calculus (Pure)

- Simple structure
- No side-effects
- Easy to reason about
- Useless for programmers(?)

Real (Impure)

- Complicated structure
- Side-effects
- Hard to reason about
- Interact with the real world

Programming Languages

Is the λ-calculus *useless* for programmers?

Programming Languages

Is the λ-calculus *useless* for programmers?

[The correspondence] reduces the problem of specifying ALGOL 60 semantics to that of specifying the semantics of a structurally simpler language.

Peter Landin

in "Correspondence between ALGOL 60 and Church's Lambda-notation: part I"

Programming Languages

Is the λ-calculus *useless* for programmers?

[The correspondence] reduces the problem of specifying ALGOL 60 semantics to that of specifying the semantics of a structurally simpler language.

Peter Landin

in "Correspondence between ALGOL 60 and Church's Lambda-notation: part I"

How can we add *effects* to pure languages?

Programming Languages

Is the λ-calculus *useless* for programmers?

[The correspondence] reduces the problem of specifying ALGOL 60 semantics to that of specifying the semantics of a structurally simpler language.

Peter Landin

in "Correspondence between ALGOL 60 and Church's Lambda-notation: part I"

How can we add *effects* to pure languages?

[W]e distinguish the object A of values (of type A) from the object TA of computations (of type A).

Eugenio Moggi

in "Notions of Computation and Monads"

Exceptions




Exceptions

Moggi's CBV Encoding

Effect Operations

Exceptions

Moggi's CBV Encoding

Let E be the type of exceptions.

Then $TA = (\mathcal{E} \oplus A)$:

Effect Operations

Exceptions

Moggi's CBV Encoding

Let E be the type of exceptions.

Then $TA = (\mathcal{E} \oplus A)$:

$$v \quad \rightsquigarrow \quad v$$

Effect Operations

Exceptions

Moggi's CBV Encoding

Let E be the type of exceptions.

Then $TA = (\mathcal{E} \oplus A)$:

$$v \quad \rightsquigarrow \quad \mathsf{in}_r(v)$$

Effect Operations

Exceptions

Moggi's CBV Encoding

Let E be the type of exceptions.

Then $TA = (\mathcal{E} \oplus A)$:

$$
\begin{array}{rcl}
v & \rightsquigarrow & \mathsf{in}_r(v) \\
t\ u & \rightsquigarrow & \qquad u \\
& & \qquad\qquad t
\end{array}
$$

Effect Operations

Exceptions

Moggi's CBV Encoding

Let E be the type of exceptions.

Then $TA = (\mathcal{E} \oplus A)$:

$$\begin{aligned} v &\rightsquigarrow \mathsf{in}_r(v) \\ t\ u &\rightsquigarrow \mathsf{case}\ u\ \mathsf{of}\ \mathsf{in}_l(e) \mapsto \\ &\qquad\qquad \mathsf{in}_r(v) \mapsto t \end{aligned}$$

Effect Operations

Exceptions

Moggi's CBV Encoding

Let E be the type of exceptions.

Then $TA = (\mathcal{E} \oplus A)$:

$$
\begin{array}{rcl}
v & \rightsquigarrow & \mathsf{in}_r(v) \\
t\ u & \rightsquigarrow & \mathsf{case}\ u\ \mathsf{of}\ \mathsf{in}_l(e) \mapsto \mathsf{in}_l(e) \\
 & & \qquad\qquad\quad \mathsf{in}_r(v) \mapsto t
\end{array}
$$

Effect Operations

Exceptions

Moggi's CBV Encoding

Let E be the type of exceptions.

Then $TA = (\mathcal{E} \oplus A)$:

$$
\begin{array}{rcl}
v & \rightsquigarrow & \mathsf{in}_r(v) \\
t\ u & \rightsquigarrow & \mathsf{case}\ u\ \mathsf{of}\ \mathsf{in}_l(e) \mapsto \mathsf{in}_l(e) \\
 & & \qquad\qquad\quad \mathsf{in}_r(v) \mapsto t\ v
\end{array}
$$

Effect Operations

Exceptions

Moggi's CBV Encoding

Let E be the type of exceptions.

Then $TA = (\mathcal{E} \oplus A)$:

$$
\begin{array}{rcl}
v & \rightsquigarrow & \mathsf{in}_r(v) \\
t\ u & \rightsquigarrow & \mathsf{case}\ u\ \mathsf{of}\ \mathsf{in}_l(e) \mapsto \mathsf{in}_l(e) \\
 & & \qquad\qquad\quad \mathsf{in}_r(v) \mapsto t\ v
\end{array}
$$

Effect Operations

Let e be an exception name:

Exceptions

Moggi's CBV Encoding

Let E be the type of exceptions.

Then $TA = (\mathcal{E} \oplus A)$:

$$\begin{array}{rcl} v & \rightsquigarrow & \mathsf{in}_r(v) \\ t\ u & \rightsquigarrow & \mathsf{case}\ u\ \mathsf{of}\ \mathsf{in}_l(e) \mapsto \mathsf{in}_l(e) \\ & & \qquad\qquad \mathsf{in}_r(v) \mapsto t\ v \end{array}$$

Effect Operations

Let e be an exception name:

- Raising an exception:

$$\mathtt{raise}_e()$$

Exceptions

Moggi's CBV Encoding

Let E be the type of exceptions.

Then $TA = (\mathcal{E} \oplus A)$:

$$
\begin{array}{rcl}
v & \rightsquigarrow & \mathsf{in}_r(v) \\
t\ u & \rightsquigarrow & \mathsf{case}\ u\ \mathsf{of}\ \mathsf{in}_l(e) \mapsto \mathsf{in}_l(e) \\
 & & \qquad\qquad \mathsf{in}_r(v) \mapsto t\ v
\end{array}
$$

Effect Operations

Let e be an exception name:

- Raising an exception:

$$\mathtt{raise}_e()$$

- Handling an exception:

$$\mathtt{handle}_e(t, u)$$

Non-Idempotent Intersection Types




Non-Idempotent Intersection Types

- Intersection types that do not enjoy idempotency $(\tau \cap \tau) \neq \tau$

Non-Idempotent Intersection Types

- Intersection types that do not enjoy idempotency $(\tau \cap \tau) \neq \tau$
- Express models capturing upper bound quantitative computational properties

"t is terminating in at most X steps iff t is typable"

Non-Idempotent Intersection Types

- Intersection types that do not enjoy idempotency $(\tau \cap \tau) \neq \tau$
- Express models capturing upper bound quantitative computational properties

"t is terminating in at most X steps iff t is typable"

- Size of type derivations is an upper bound for

evaluation length + size of result

Non-Idempotent Intersection Types

- Intersection types that do not enjoy idempotency $(\tau \cap \tau) \neq \tau$
- Express models capturing upper bound quantitative computational properties

"t is terminating in at most X steps iff t is typable"

- Size of type derivations is an upper bound for

evaluation length + size of result

- Size explosion

$$\begin{array}{lcl} t_0 & := & y \\ t_n & := & (\lambda x.xx)t_{n-1} \end{array} \quad \rightsquigarrow \quad t_n \to_\beta^n y^{2^n}$$

Non-Idempotent Intersection Types

- Intersection types that do not enjoy idempotency $(\tau \cap \tau) \neq \tau$
- Express models capturing upper bound quantitative computational properties

"t is terminating in at most X steps iff t is typable"

- Size of type derivations is an upper bound for

evaluation length + size of result

- Size explosion

$$\begin{array}{lcl} t_0 & := & y \\ t_n & := & (\lambda x.xx)t_{n-1} \end{array} \quad \rightsquigarrow \quad \underbrace{t_n}_{\text{linear in } n} \to_\beta^n y^{2^n}$$

Non-Idempotent Intersection Types

- Intersection types that do not enjoy idempotency $(\tau \cap \tau) \neq \tau$
- Express models capturing upper bound quantitative computational properties

"t is terminating in at most X steps iff t is typable"

- Size of type derivations is an upper bound for

evaluation length + size of result

- Size explosion

$$\begin{array}{lcl} t_0 & := & y \\ t_n & := & (\lambda x.xx)t_{n-1} \end{array} \quad \rightsquigarrow \quad \underbrace{t_n}_{\text{linear in } n} \; \overbrace{\rightarrow_\beta^n \; y^{2^n}}^{\text{exponential in } n}$$

Split and Exact Measures

- To obtain split measures

- To obtain exact measures

Split and Exact Measures

- To obtain split measures

counters in judgments + tight constants + persistent typing rules

- To obtain exact measures

Split and Exact Measures

- To obtain split measures

$$\underbrace{\text{counters in judgments} + \text{tight constants} + \text{persistent typing rules}}_{\text{(evaluation length, size of result)}}$$

- To obtain exact measures

Split and Exact Measures

- To obtain split measures

$$\underbrace{\text{counters in judgments} + \text{tight constants} + \text{persistent typing rules}}_{\text{(evaluation length, size of result)}}$$

- To obtain exact measures

$$\text{tight derivations} = \text{minimal derivations}$$

Split and Exact Measures

- To obtain split measures

$$\underbrace{\text{counters in judgments} + \text{tight constants} + \text{persistent typing rules}}$$

(evaluation length, size of result)

- To obtain exact measures

tight derivations = minimal derivations

- Obtain models capturing exact quantitative computational properties

"t is terminating in exactly X steps with normal form of size Y
iff t is typable with counter (X, Y)"

A Quantitative Understanding of Exceptions

Goal

To build a quantitative model (expressed as a tight type system) that captures exact quantitative properties of a λ-calculus with operations that raise and handle exceptions.

Syntax

$$|\mathtt{raise}_e()| = 1 \quad |x| = 0 \quad |\lambda x.t| = 0 \quad |vt| = 1 + |t| \quad |\mathtt{handle}_e(t,u)| = 1 + |t|$$

Syntax

- We distinguish between and

Values	$::=$
Terms	$::=$

$$|\mathtt{raise}_e()| = 1 \quad |x| = 0 \quad |\lambda x.t| = 0 \quad |vt| = 1 + |t| \quad |\mathtt{handle}_e(t,u)| = 1 + |t|$$

Syntax

- We distinguish between values v and

Values	v, w	$::=$	$x \mid \lambda x.t$
Terms		$::=$	

$$|\mathtt{raise}_e()| = 1 \quad |x| = 0 \quad |\lambda x.t| = 0 \quad |vt| = 1 + |t| \quad |\mathtt{handle}_e(t, u)| = 1 + |t|$$

Syntax

- We distinguish between values v and computations t (terms)

$$\begin{array}{llll} \text{Values} & v, w & ::= & x \mid \lambda x.t \\ \text{Terms} & t, u & ::= & v \mid \ldots \end{array}$$

$$|\mathtt{raise}_e()| = 1 \quad |x| = 0 \quad |\lambda x.t| = 0 \quad |vt| = 1 + |t| \quad |\mathtt{handle}_e(t, u)| = 1 + |t|$$

Syntax

- We distinguish between values v and computations t (terms)
- Applications are restricted to the form vt

$$\begin{array}{llll} \text{Values} & v, w & ::= & x \mid \lambda x.t \\ \text{Terms} & t, u & ::= & v \mid vt \mid \dots \end{array}$$

$$|\mathtt{raise}_e()| = 1 \quad |x| = 0 \quad |\lambda x.t| = 0 \quad |vt| = 1 + |t| \quad |\mathtt{handle}_e(t, u)| = 1 + |t|$$

Syntax

- We distinguish between values v and computations t (terms)
- Applications are restricted to the form vt
- Effect operations are used to raise and handle exceptions

$$\begin{array}{llll} \text{Values} & v, w & ::= & x \mid \lambda x.t \\ \text{Terms} & t, u & ::= & v \mid vt \mid \mathtt{raise}_e() \mid \mathtt{handle}_e(t, u) \end{array}$$

$$|\mathtt{raise}_e()| = 1 \quad |x| = 0 \quad |\lambda x.t| = 0 \quad |vt| = 1 + |t| \quad |\mathtt{handle}_e(t, u)| = 1 + |t|$$

Operational Semantics

Operational Semantics

$$\frac{}{(\lambda x.t)v \to t\{x \backslash v\}}\ (\mathrm{b})$$

Operational Semantics

$$\frac{}{(\lambda x.t)v \to t\{x \backslash v\}}\ (\mathrm{b}) \qquad \frac{}{v\ \mathtt{raise}_e() \to \mathtt{raise}_e()}\ (\mathrm{r})$$

Operational Semantics

$$\frac{}{(\lambda x.t)v \to t\{x \backslash v\}}\ (\mathtt{b}) \qquad \frac{}{v\ \mathtt{raise}_e() \to \mathtt{raise}_e()}\ (\mathtt{r})$$

$$\frac{}{\mathtt{handle}_e(\mathtt{raise}_e(), t) \to t}\ (\mathtt{h1})$$

Operational Semantics

$$\frac{}{(\lambda x.t)v \to t\{x \backslash v\}}\ (\mathrm{b}) \qquad \frac{}{v\ \mathtt{raise}_e() \to \mathtt{raise}_e()}\ (\mathrm{r})$$

$$\frac{}{\mathtt{handle}_e(\mathtt{raise}_e(), t) \to t}\ (\mathtt{h1}) \qquad \frac{[\mathtt{isvalue}(t)] \text{ or } [\mathtt{israise}_{e'}(t)\ (e' \neq e)]}{\mathtt{handle}_e(t, u) \to t}\ (\mathtt{h2})$$

Operational Semantics

$$\frac{}{(\lambda x.t)v \to t\{x \backslash v\}}\ (\mathrm{b}) \qquad \frac{}{v\ \mathtt{raise}_e() \to \mathtt{raise}_e()}\ (\mathrm{r})$$

$$\frac{}{\mathtt{handle}_e(\mathtt{raise}_e(), t) \to t}\ (\mathtt{h1}) \qquad \frac{[\mathtt{isvalue}(t)] \text{ or } [\mathtt{israise}_{e'}(t)\ (e' \neq e)]}{\mathtt{handle}_e(t, u) \to t}\ (\mathtt{h2})$$

$$\frac{t \to u}{\mathtt{handle}_e(t, p) \to \mathtt{handle}_e(u, p)}\ (\mathtt{c1}) \qquad \frac{t \to u}{vt \to vu}\ (\mathtt{c2})$$

Operational Semantics

$$\frac{}{(\lambda x.t)v \to t\{x \backslash v\}}\ (\mathrm{b}) \qquad \frac{}{v\ \mathtt{raise}_e() \to \mathtt{raise}_e()}\ (\mathrm{r})$$

$$\frac{}{\mathtt{handle}_e(\mathtt{raise}_e(), t) \to t}\ (\mathtt{h1}) \qquad \frac{[\mathtt{isvalue}(t)] \text{ or } [\mathtt{israise}_{e'}(t)\ (e' \neq e)]}{\mathtt{handle}_e(t, u) \to t}\ (\mathtt{h2})$$

$$\frac{t \to u}{\mathtt{handle}_e(t, p) \to \mathtt{handle}_e(u, p)}\ (\mathtt{c1}) \qquad \frac{t \to u}{vt \to vu}\ (\mathtt{c2})$$

Weak reduction: we do not reduce inside abstractions

We allow *open* normal forms

Operational Semantics ⚜ Example

$$(\lambda x.\mathtt{handle}_e(xy, x))\ (\lambda z.\mathtt{raise}_e())$$

(0 # b-steps, 0 # exceptions handled, 0# exceptions propagated)

Operational Semantics ⚜ Example

$$(\lambda x.\texttt{handle}_e(xy, x))\ \boxed{(\lambda z.\texttt{raise}_e())}$$

(0 # b-steps, 0 # exceptions handled, 0# exceptions propagated)

Operational Semantics ⚜ Example

$$
\begin{aligned}
& (\lambda x.\mathtt{handle}_e(xy, x))\ \boxed{(\lambda z.\mathtt{raise}_e())} \\
\to_{\mathtt{b}}\ & \mathtt{handle}_e((\lambda z.\mathtt{raise}_e())\ y, (\lambda z.\mathtt{raise}_e()))
\end{aligned}
$$

(1 # b-steps, 0 # exceptions handled, 0# exceptions propagated)

Operational Semantics ⚜ Example

$$
\begin{aligned}
& (\lambda x.\mathtt{handle}_e(xy, x))\ \boxed{(\lambda z.\mathtt{raise}_e())} \\
\to_{\mathtt{b}}\ & \mathtt{handle}_e((\lambda z.\mathtt{raise}_e())\ \boxed{y}, (\lambda z.\mathtt{raise}_e()))
\end{aligned}
$$

(1 # b-steps, 0 # exceptions handled, 0# exceptions propagated)

Operational Semantics ⚜ Example

$$
\begin{aligned}
& (\lambda x.\mathtt{handle}_e(xy, x))\ \boxed{(\lambda z.\mathtt{raise}_e())} \\
\to_{\mathtt{b}}\ & \mathtt{handle}_e((\lambda z.\mathtt{raise}_e())\ \boxed{y}, (\lambda z.\mathtt{raise}_e())) \\
\to_{\mathtt{b}}\ & \mathtt{handle}_e(\mathtt{raise}_e(), \lambda z.\mathtt{raise}_e())
\end{aligned}
$$

(2 # b-steps, 0 # exceptions handled, 0# exceptions propagated)

Operational Semantics ⚜ Example

$$
\begin{aligned}
& (\lambda x.\mathtt{handle}_e(xy, x))\ \boxed{(\lambda z.\mathtt{raise}_e())} \\
\to_{\mathtt{b}}\ & \mathtt{handle}_e((\lambda z.\mathtt{raise}_e())\ \boxed{y}, (\lambda z.\mathtt{raise}_e())) \\
\to_{\mathtt{b}}\ & \mathtt{handle}_{\boxed{e}}(\mathtt{raise}_{\boxed{e}}(), \lambda z.\mathtt{raise}_e())
\end{aligned}
$$

(2 # b-steps, 0 # exceptions handled, 0# exceptions propagated)

Operational Semantics ⚜ Example

$$
\begin{aligned}
& (\lambda x.\mathtt{handle}_e(xy, x))\ (\lambda z.\mathtt{raise}_e()) \\
\to_{\mathtt{b}}\ & \mathtt{handle}_e((\lambda z.\mathtt{raise}_e())\ y, (\lambda z.\mathtt{raise}_e())) \\
\to_{\mathtt{b}}\ & \mathtt{handle}_e(\mathtt{raise}_e(), \lambda z.\mathtt{raise}_e()) \\
\to_{\mathtt{h1}}\ & \lambda z.\mathtt{raise}_e()
\end{aligned}
$$

(2 # b-steps, 1 # exceptions handled, 0# exceptions propagated)

Encoding Arrow Types

Encoding Arrow Types

$$\underbrace{A \Rightarrow B}_{\text{IL}}$$

Encoding Arrow Types

$$\underbrace{A \Rightarrow B}_{\text{IL}} \overset{\text{Girard's CBV}}{\rightsquigarrow} \underbrace{!A \multimap !B}_{\text{ILL}}$$

Encoding Arrow Types

$$\underbrace{A \Rightarrow B}_{\text{IL}} \overset{\text{Girard's CBV}}{\rightsquigarrow} \underbrace{!A \multimap !B}_{\text{ILL}} \overset{\text{Moggi's CBV}}{\rightsquigarrow} !A \multimap T(!B)$$

Encoding Arrow Types

$$\underbrace{A \Rightarrow B}_{\text{IL}} \overset{\text{Girard's CBV}}{\rightsquigarrow} \underbrace{!A \multimap !B}_{\text{ILL}} \overset{\text{Moggi's CBV}}{\rightsquigarrow} !A \multimap T(!B)$$

- $!A$ is an intersection of value types

$$!A = [A, \dots, A]$$

Encoding Arrow Types

$$\underbrace{A \Rightarrow B}_{\text{IL}} \overset{\text{Girard's CBV}}{\rightsquigarrow} \underbrace{!A \multimap !B}_{\text{ILL}} \overset{\text{Moggi's CBV}}{\rightsquigarrow} !A \multimap T(!B)$$

- $!A$ is an intersection of value types

$$!A = [A, \dots, A]$$

- T is the exceptions monad

$$TA = \mathcal{E} \oplus A$$

Encoding Arrow Types

$$\underbrace{A \Rightarrow B}_{\text{IL}} \overset{\text{Girard's CBV}}{\rightsquigarrow} \underbrace{!A \multimap !B}_{\text{ILL}} \overset{\text{Moggi's CBV}}{\rightsquigarrow} !A \multimap T(!B)$$

- $!A$ is an intersection of value types

$$!A = [A, \dots, A]$$

- T is the exceptions monad

$$TA = \mathcal{E} \oplus A$$

- $T(!A)$ is a computation wrapping an intersection of value types

$$T[A, \dots, A] = \mathcal{E} \oplus [A, \dots, A]$$

Types

Types

- Values and Neutral Forms

- Computations

Types

- Values and Neutral Forms

Tight Constants	::=
Value Types	::=
Multi-types	::=
Liftable Types	::=
Types	::=

- Computations

Types

- Values and Neutral Forms

Tight Constants	$\mathtt{tt}$	$::=$	$\mathtt{v} \mid \mathtt{a} \mid \mathtt{n}$
Value Types		$::=$	
Multi-types		$::=$	
Liftable Types		$::=$	
Types		$::=$	

- Computations

Types

- Values and Neutral Forms

Tight Constants	$\mathtt{tt}$	$::=$	$\mathtt{v} \mid \mathtt{a} \mid \mathtt{n}$
Value Types	σ	$::=$	$\mathtt{v} \mid \mathtt{a} \mid \mathcal{M} \Rightarrow \delta$
Multi-types		$::=$	
Liftable Types		$::=$	
Types		$::=$	

- Computations

Types

- Values and Neutral Forms

Tight Constants	$\mathtt{tt}$	$::=$	$\mathtt{v} \mid \mathtt{a} \mid \mathtt{n}$
Value Types	σ	$::=$	$\mathtt{v} \mid \mathtt{a} \mid \mathcal{M} \Rightarrow \delta$
Multi-types	$\mathcal{M}$	$::=$	$[\sigma_i]_{i \in I}$ where I is a finite set
Liftable Types		$::=$	
Types		$::=$	

- Computations

Types

- Values and Neutral Forms

Tight Constants	$\mathtt{tt}$	$::=$	$\mathtt{v} \mid \mathtt{a} \mid \mathtt{n}$
Value Types	σ	$::=$	$\mathtt{v} \mid \mathtt{a} \mid \mathcal{M} \Rightarrow \delta$
Multi-types	$\mathcal{M}$	$::=$	$[\sigma_i]_{i \in I}$ where I is a finite set
Liftable Types	μ	$::=$	$\mathtt{v} \mid \mathtt{a} \mid \mathcal{M}$
Types		$::=$	

- Computations

Types

- Values and Neutral Forms

Tight Constants	$\mathtt{tt}$	$::=$	$\mathtt{v} \mid \mathtt{a} \mid \mathtt{n}$
Value Types	σ	$::=$	$\mathtt{v} \mid \mathtt{a} \mid \mathcal{M} \Rightarrow \delta$
Multi-types	$\mathcal{M}$	$::=$	$[\sigma_i]_{i \in I}$ where I is a finite set
Liftable Types	μ	$::=$	$\mathtt{v} \mid \mathtt{a} \mid \mathcal{M}$
Types	τ	$::=$	$\mathtt{tt} \mid \mathcal{M}$

- Computations

Types

- Values and Neutral Forms

Tight Constants	$\mathtt{tt}$	$::=$	$\mathtt{v} \mid \mathtt{a} \mid \mathtt{n}$
Value Types	σ	$::=$	$\mathtt{v} \mid \mathtt{a} \mid \mathcal{M} \Rightarrow \delta$
Multi-types	$\mathcal{M}$	$::=$	$[\sigma_i]_{i \in I}$ where I is a finite set
Liftable Types	μ	$::=$	$\mathtt{v} \mid \mathtt{a} \mid \mathcal{M}$
Types	τ	$::=$	$\mathtt{tt} \mid \mathcal{M}$

- Computations

Monadic Types	$::=$

Types

- Values and Neutral Forms

Tight Constants	$\mathtt{tt}$	$::=$	$\mathtt{v} \mid \mathtt{a} \mid \mathtt{n}$
Value Types	σ	$::=$	$\mathtt{v} \mid \mathtt{a} \mid \mathcal{M} \Rightarrow \delta$
Multi-types	$\mathcal{M}$	$::=$	$[\sigma_i]_{i \in I}$ where I is a finite set
Liftable Types	μ	$::=$	$\mathtt{v} \mid \mathtt{a} \mid \mathcal{M}$
Types	τ	$::=$	$\mathtt{tt} \mid \mathcal{M}$

- Computations

Monadic Types	δ	$::=$	$\gamma \oplus \tau$ where $\gamma \in \mathcal{E} \cup \{\star\}$

Type System

Type System

- Judgments are decorated with counters

$$(b \quad , \quad h \quad , \quad p \quad , \quad s)$$

Type System

- Judgments are decorated with counters

$$\overbrace{(b}^{\#\ \beta\text{-steps}}\ ,\quad h\ ,\quad p\ ,\quad s)$$

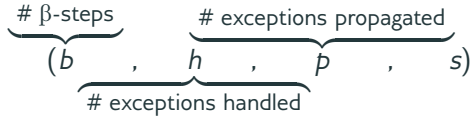
Type System

- Judgments are decorated with counters

$$\overbrace{(b}^{\#\ \beta\text{-steps}}\ ,\ \overbrace{h}^{\#\ \text{exceptions handled}}\ ,\ p\ ,\ s)$$

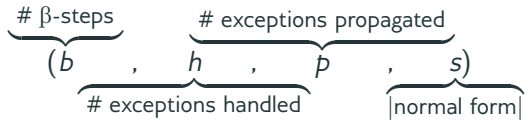
Type System

- Judgments are decorated with counters



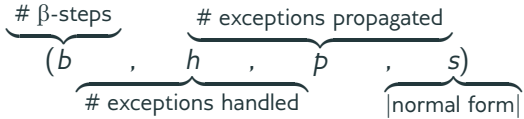
Type System

- Judgments are decorated with counters



Type System

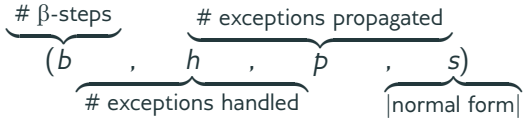
- Judgments are decorated with counters



- Some typing rules have two (or more) different versions

Type System

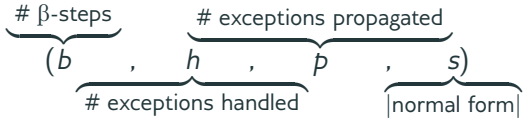
- Judgments are decorated with counters



- Some typing rules have two (or more) different versions
  - *Consuming*: increase only *b*, *h*, and *p* counters

Type System

- Judgments are decorated with counters



- Some typing rules have two (or more) different versions
  - *Consuming*: increase only *b*, *h*, and *p* counters
  - *Persistent*: increase the *s* counter

(Some) Typing Rules

(Some) Typing Rules

$$\frac{(\Gamma_i \vdash^{(b_i,h_i,p_i,s_i)} v : \sigma_i)_{i\in I}}{+_{i\in I}\,\Gamma_i \vdash^{(+_{i\in I}b_i,+_{i\in I}h_i,+_{i\in I}p_i,+_{i\in I}s_i)} v : [\sigma_i]_{i\in I}}\ (\texttt{many})$$

(Some) Typing Rules

$$\frac{(\Gamma_i \vdash^{(b_i,h_i,p_i,s_i)} v : \sigma_i)_{i\in I}}{+_{i\in I}\,\Gamma_i \vdash^{(+_{i\in I}b_i,+_{i\in I}h_i,+_{i\in I}p_i,+_{i\in I}s_i)} v : [\sigma_i]_{i\in I}}\ (\texttt{many}) \qquad \frac{\Gamma \vdash^{(b,h,p,s)} v : \mu}{\Gamma \vdash^{(b,h,p,s)} v : \star \oplus \mu}\ (\texttt{unit})$$

(Some) Typing Rules

$$\frac{(\Gamma_i \vdash^{(b_i,h_i,p_i,s_i)} v : \sigma_i)_{i\in I}}{+_{i\in I}\,\Gamma_i \vdash^{(+_{i\in I}b_i,+_{i\in I}h_i,+_{i\in I}p_i,+_{i\in I}s_i)} v : [\sigma_i]_{i\in I}}\ (\texttt{many}) \qquad \frac{\Gamma \vdash^{(b,h,p,s)} v : \mu}{\Gamma \vdash^{(b,h,p,s)} v : \star \oplus \mu}\ (\texttt{unit})$$

$$\frac{\Gamma \vdash^{(b,h,p,s)} v : \mathcal{M} \Rightarrow \delta \qquad \Delta \vdash^{(b',h',p',s')} t : \star \oplus \mathcal{M}}{\Gamma + \Delta \vdash^{(1+b+b',h+h',p+p',s+s')} vt : \delta}\ (\text{app})$$

(Some) Typing Rules

$$\frac{(\Gamma_i \vdash^{(b_i,h_i,p_i,s_i)} v : \sigma_i)_{i\in I}}{+_{i\in I}\,\Gamma_i \vdash^{(+_{i\in I}b_i,+_{i\in I}h_i,+_{i\in I}p_i,+_{i\in I}s_i)} v : [\sigma_i]_{i\in I}}\ (\texttt{many}) \qquad \frac{\Gamma \vdash^{(b,h,p,s)} v : \mu}{\Gamma \vdash^{(b,h,p,s)} v : \star \oplus \mu}\ (\texttt{unit})$$

$$\frac{\Gamma \vdash^{(b,h,p,s)} v : \mathcal{M} \Rightarrow \delta \qquad \Delta \vdash^{(b',h',p',s')} t : \star \oplus \mathcal{M}}{\Gamma + \Delta \vdash^{(1+b+b',h+h',p+p',s+s')} vt : \delta}\ (\text{app})$$

$$\frac{\Gamma \vdash^{(b,h,p,s)} t : e \oplus [] \qquad \Delta \vdash^{(b',h',p',s')} u : \delta}{\Gamma + \Delta \vdash^{(b+b',1+h+h',p+p',s+s')} \texttt{handle}_e(t,u) : \delta}\ (\texttt{handle1})$$

Exact Measures ( )

Why do we need *tightness* and *persistent* typing rules?

Exact Measures ( )

Why do we need *tightness* and *persistent* typing rules?

Let $\sigma = [\mathrm{v}] \Rightarrow (\gamma \oplus \tau)$.

$$\dfrac{\dfrac{}{x : [\sigma] \vdash^{(0,0,0,0)} x : [\mathrm{v}] \Rightarrow (\gamma \oplus \tau)}\,(\mathtt{ax}) \qquad \dfrac{\dfrac{\dfrac{}{y : [\mathrm{v}] \vdash^{(0,0,0,0)} y : \mathrm{v}}\,(\mathtt{ax})}{y : [\mathrm{v}] \vdash^{(0,0,0,0)} y : [\mathrm{v}]}\,(\mathtt{many})}{y : [\mathrm{v}] \vdash^{(0,0,0,0)} y : \star \oplus [\mathrm{v}]}\,(\mathtt{unit})}{x : [\sigma], y : [\mathrm{v}] \vdash^{(1,0,0,0)} xy : \gamma \oplus \tau}\,(\mathtt{app})$$

Exact Measures ( )

Why do we need *tightness* and *persistent* typing rules?

Let $\sigma = [\mathrm{v}] \Rightarrow (\gamma \oplus \tau)$.

$$\dfrac{\dfrac{}{x : [\sigma] \vdash^{(0,0,0,0)} x : [\mathrm{v}] \Rightarrow (\gamma \oplus \tau)}\,(\mathtt{ax}) \qquad \dfrac{\dfrac{\dfrac{}{y : [\mathrm{v}] \vdash^{(0,0,0,0)} y : \mathrm{v}}\,(\mathtt{ax})}{y : [\mathrm{v}] \vdash^{(0,0,0,0)} y : [\mathrm{v}]}\,(\mathtt{many})}{y : [\mathrm{v}] \vdash^{(0,0,0,0)} y : \star \oplus [\mathrm{v}]}\,(\mathtt{unit})}{x : [\sigma], y : [\mathrm{v}] \vdash^{(1,0,0,0)} xy : \gamma \oplus \tau}\,(\mathtt{app})$$

$$\underbrace{|\mathrm{xy}| = 1}_{xy} \not\to$$

Exact Measures (Wrong)

Why do we need *tightness* and *persistent* typing rules?

Let $\sigma = [\mathrm{v}] \Rightarrow (\gamma \oplus \tau)$.

$$\dfrac{\dfrac{}{x : [\sigma] \vdash^{(0,0,0,0)} x : [\mathrm{v}] \Rightarrow (\gamma \oplus \tau)}\,(\mathtt{ax}) \qquad \dfrac{\dfrac{\dfrac{}{y : [\mathrm{v}] \vdash^{(0,0,0,0)} y : \mathrm{v}}\,(\mathtt{ax})}{y : [\mathrm{v}] \vdash^{(0,0,0,0)} y : [\mathrm{v}]}\,(\mathtt{many})}{y : [\mathrm{v}] \vdash^{(0,0,0,0)} y : \star \oplus [\mathrm{v}]}\,(\mathtt{unit})}{x : [\sigma], y : [\mathrm{v}] \vdash^{(1,0,0,0)} xy : \gamma \oplus \tau}\,(\mathtt{app})$$

$$\underbrace{|\mathrm{xy}| = 1}_{xy} \not\to$$

(Some) Typing Rules ⚜ Persistent

$$\frac{}{\vdash^{(0,0,0,0)} \lambda x.t : \mathtt{a}}\ (\mathrm{abs}_p)$$

$$\frac{\Gamma \vdash^{(b,h,p,s)} t : \star \oplus \overline{\mathtt{r}}}{x : [\mathtt{v}] + \Gamma \vdash^{(b,h,p,1+s)} xt : \star \oplus \mathtt{n}}\ (\mathrm{app1}_p)$$

$$\frac{\Gamma \vdash^{(b,h,p,s)} t : \star \oplus \mathtt{n}}{\Gamma \vdash^{(b,h,p,1+s)} (\lambda x.u)t : \star \oplus \mathtt{n}}\ (\mathrm{app2}_p)$$

Exact Measures ( )

Exact Measures ( )

$$\dfrac{\dfrac{\dfrac{}{y : [\mathtt{v}] \vdash^{(0,0,0,0)} y : \mathtt{v}}\,(\mathtt{ax})}{y : [\mathtt{v}] \vdash^{(0,0,0,0)} y : \star \oplus \mathtt{v}}\,(\mathtt{unit})}{x : [\mathtt{v}], y : [\mathtt{v}] \vdash^{(0,0,0,1)} xy : \star \oplus \mathtt{n}}\,(\mathtt{app1}_p)$$

Exact Measures ( )

$$\dfrac{\dfrac{\dfrac{}{y : [\mathtt{v}] \vdash^{(0,0,0,0)} y : \mathtt{v}}\,(\mathtt{ax})}{y : [\mathtt{v}] \vdash^{(0,0,0,0)} y : \star \oplus \mathtt{v}}\,(\mathtt{unit})}{x : [\mathtt{v}], y : [\mathtt{v}] \vdash^{(0,0,0,1)} xy : \star \oplus \mathtt{n}}\,(\mathtt{app1}_p)$$

$$\underbrace{|\mathsf{xy}| = 1}_{xy} \not\to$$

Exact Measures (Correct)

$$\dfrac{\dfrac{\dfrac{}{y : [\mathtt{v}] \vdash^{(0,0,0,0)} y : \mathtt{v}}\,(\mathtt{ax})}{y : [\mathtt{v}] \vdash^{(0,0,0,0)} y : \star \oplus \mathtt{v}}\,(\mathtt{unit})}{x : [\mathtt{v}], y : [\mathtt{v}] \vdash^{(0,0,0,1)} xy : \star \oplus \mathtt{n}}\,(\mathtt{app1}_p)$$

$$\underbrace{|\mathtt{xy}| = 1}_{xy} \not\to$$

Validity of the Model

Soundness

Completeness

Validity of the Model

Soundness

If $\Phi \triangleright \Gamma \vdash^{(b,h,p,s)} t : \delta$ tight,

Completeness

Validity of the Model

Soundness

If $\Phi \triangleright \Gamma \vdash^{(b,h,p,s)} t : \delta$ tight,
then $\exists u \in \mathtt{no}$, s.t. $t \twoheadrightarrow^{(b,h,p)} u$, and $|u| = s$.

Completeness

Validity of the Model

Soundness

If $\Phi \triangleright \Gamma \vdash^{(b,h,p,s)} t : \delta$ tight,
then $\exists u \in \mathtt{no}$, s.t. $t \twoheadrightarrow^{(b,h,p)} u$, and $|u| = s$.

Completeness

Validity of the Model

Soundness

If $\Phi \triangleright \Gamma \vdash^{(b,h,p,s)} t : \delta$ tight,
then $\exists u \in \mathtt{no}$, s.t. $t \twoheadrightarrow^{(b,h,p)} u$, and $|u| = s$.

Completeness

If $t \twoheadrightarrow^{(b,h,p)} u$,

Validity of the Model

Soundness

If $\Phi \triangleright \Gamma \vdash^{(b,h,p,s)} t : \delta$ tight,
then $\exists u \in \mathtt{no}$, s.t. $t \twoheadrightarrow^{(b,h,p)} u$, and $|u| = s$.

Completeness

If $t \twoheadrightarrow^{(b,h,p)} u$,
then $\exists \Phi \triangleright \Gamma \vdash^{(b,h,p,|u|)} t : \delta$ tight.

Typing Example

Let us consider the term exemplifying the operational semantics:

$$(\lambda x.\mathtt{handle}_e(xy, x))\ (\lambda z.\mathtt{raise}_e())\ \twoheadrightarrow^{(2,1,0)}\ \underbrace{\mid \lambda z.\mathtt{raise}_e() \mid = 0}_{\lambda z.\mathtt{raise}_e()}$$

Typing Example

- Let us consider the term exemplifying the operational semantics:

$$(\lambda x.\mathtt{handle}_e(xy, x))\ (\lambda z.\mathtt{raise}_e())\ \twoheadrightarrow^{(2,1,0)}\ \underbrace{|\ \lambda z.\mathtt{raise}_e()\ |\ =\ 0}_{\lambda z.\mathtt{raise}_e()}$$

Typing Example

- Let us consider the term exemplifying the operational semantics:

$$(\lambda x.\mathtt{handle}_e(xy, x))\ (\lambda z.\mathtt{raise}_e())\ \twoheadrightarrow^{(2,1,0)}\ \underbrace{\mid \lambda z.\mathtt{raise}_e() \mid = 0}_{\lambda z.\mathtt{raise}_e()}$$

- We can build the following tight derivation:

$$\frac{\Phi \qquad \Psi}{y : [] \vdash^{(2,1,0,0)} (\lambda x.\mathtt{handle}_e(xy, x))(\lambda z.\mathtt{raise}_{e'}()) : \star \oplus \mathtt{a}}\ (\mathrm{app})$$

Conclusions

Summary

- Simple language capable of raising and handling exceptions
- Following a weak (open) CBV strategy
- Provided a quantitative model capturing exact measures

Conclusions

Summary

- Simple language capable of raising and handling exceptions
- Following a weak (open) CBV strategy
- Provided a quantitative model capturing exact measures

Future Work

- Different effects: global memory, I/O, non-determinism, ...
- Different Strategies: CBV (unrestricted), CBN, CBNeed, ...
- Unifying frameworks: CBPV, λ!-calculus, EE-calculus, ...


The End