



A Quantitative Understanding of Exceptions

Miguel Ramos ¹

Riccardo Treglia ²

Delia Kesner ³

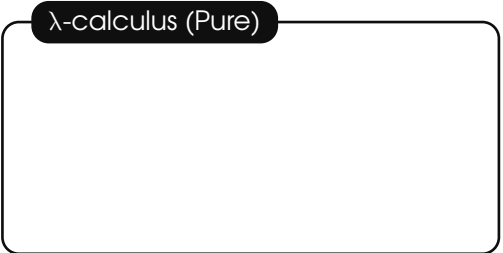

TLLA 2023

¹LIACC, DCC, Faculdade de Ciências, Universidade do Porto

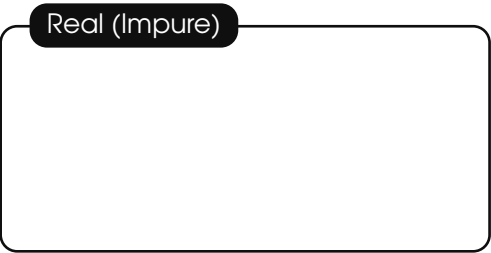

²Università di Bologna

³IRIF, 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

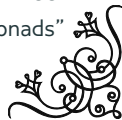
in “Correspondence between ALGOL 60 and Church’s Lambda-notation: part I”

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

$v \rightsquigarrow v$

Effect Operations

Exceptions

Moggi's CBV Encoding

Let E be the type of exceptions.

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

$$v \rightsquigarrow \text{in}_r(v)$$

Effect Operations

Exceptions

Moggi's CBV Encoding

Let E be the type of exceptions.

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

$$v \rightsquigarrow \text{in}_r(v)$$

$$t u \rightsquigarrow u$$

t

Effect Operations

Exceptions

Moggi's CBV Encoding

Let E be the type of exceptions.

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

$$v \rightsquigarrow \text{in}_r(v)$$

$$t u \rightsquigarrow \begin{array}{l} \text{case } u \text{ of } \text{in}_l(e) \mapsto \\ \text{in}_r(v) \mapsto t \end{array}$$

Effect Operations

Exceptions

Moggi's CBV Encoding

Let E be the type of exceptions.

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

$$v \rightsquigarrow \text{in}_r(v)$$

$$t \ u \rightsquigarrow \begin{array}{l} \text{case } u \text{ of } \text{in}_l(e) \mapsto \text{in}_l(e) \\ \text{in}_r(v) \mapsto t \end{array}$$

Effect Operations

Exceptions

Moggi's CBV Encoding

Let E be the type of exceptions.

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

$$v \rightsquigarrow \text{in}_r(v)$$

$$t u \rightsquigarrow \begin{array}{l} \text{case } u \text{ of } \text{in}_l(e) \mapsto \text{in}_l(e) \\ \text{in}_r(v) \mapsto t v \end{array}$$

Effect Operations

Exceptions

Moggi's CBV Encoding

Let E be the type of exceptions.

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

$$v \rightsquigarrow \text{in}_r(v)$$

$$t u \rightsquigarrow \begin{array}{l} \text{case } u \text{ of } \text{in}_l(e) \mapsto \text{in}_l(e) \\ \text{in}_r(v) \mapsto t v \end{array}$$

Effect Operations

Let e be an exception name:

Exceptions

Moggi's CBV Encoding

Let E be the type of exceptions.

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

$$v \rightsquigarrow \text{in}_r(v)$$

$$t u \rightsquigarrow \begin{array}{l} \text{case } u \text{ of } \text{in}_l(e) \mapsto \text{in}_l(e) \\ \text{in}_r(v) \mapsto t v \end{array}$$

Effect Operations

Let e be an exception name:

- Raising an exception:

`raisee()`

Exceptions

Moggi's CBV Encoding

Let E be the type of exceptions.

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

$$v \rightsquigarrow \text{in}_r(v)$$

$$t u \rightsquigarrow \begin{array}{l} \text{case } u \text{ of } \text{in}_l(e) \mapsto \text{in}_l(e) \\ \text{in}_r(v) \mapsto t v \end{array}$$

Effect Operations

Let e be an exception name:

- Raising an exception:

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

- Handling an exception:

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



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

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

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

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

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

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



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**

counters in judgments + tight constants + persistent typing rules

(evaluation length, size of result)

- To obtain **exact measures**



Split and Exact Measures



- To obtain **split measures**

counters in judgments + tight constants + persistent typing rules

(evaluation length, size of result)

- To obtain **exact measures**

tight derivations = minimal derivations



Split and Exact Measures



- To obtain **split measures**

counters in judgments + tight constants + 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





Syntax



- We distinguish between `Value` and `Term`

`Value ::=`

`Term ::=`





Syntax



- We distinguish between values v and

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

Terms $::=$





Syntax



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

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

Terms $t, u ::= v \mid \dots$





Syntax



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

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

Terms $t, u ::= v \mid vt \mid \dots$



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

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

Terms $t, u ::= v \mid vt \mid \text{raise}_e() \mid \text{handle}_e(t, u)$





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

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

Terms $t, u ::= v \mid vt \mid \text{raise}_e() \mid \text{handle}_e(t, u)$

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



Operational Semantics



A Quantitative Understanding of Exceptions

8 of 20





Operational Semantics



$$\overline{(\lambda x.t)v \rightarrow t\{x \setminus v\}} \quad \text{(b)}$$

Operational Semantics

$$\frac{}{(\lambda x.t)v \rightarrow t\{x \setminus v\}} \text{ (b)}$$

$$\frac{}{v \text{ raise}_e() \rightarrow \text{raise}_e()} \text{ (r)}$$

Operational Semantics

$$\frac{}{(\lambda x.t)v \rightarrow t\{x \setminus v\}} \text{ (b)}$$

$$\frac{}{v \text{ raise}_e() \rightarrow \text{raise}_e()} \text{ (r)}$$

$$\frac{}{\text{handle}_e(\text{raise}_e(), t) \rightarrow t} \text{ (h1)}$$

Operational Semantics

$$\frac{}{(\lambda x.t)v \rightarrow t\{x \setminus v\}} \quad \text{(b)}$$

$$\frac{}{v \text{ raise}_e() \rightarrow \text{raise}_e()} \quad \text{(r)}$$

$$\frac{}{\text{handle}_e(\text{raise}_e(), t) \rightarrow t} \quad \text{(h1)}$$

$$\frac{[\text{isvalue}(t)] \text{ or } [\text{israise}_{e'}(t) \text{ (} e' \neq e \text{)}]}{\text{handle}_e(t, u) \rightarrow t} \quad \text{(h2)}$$

Operational Semantics

$$\frac{}{(\lambda x.t)v \rightarrow t\{x \setminus v\}} \quad \text{(b)}$$

$$\frac{}{v \text{ raise}_e() \rightarrow \text{raise}_e()} \quad \text{(r)}$$

$$\frac{}{\text{handle}_e(\text{raise}_e(), t) \rightarrow t} \quad \text{(h1)}$$

$$\frac{[\text{isvalue}(t)] \text{ or } [\text{israise}_{e'}(t) \text{ (} e' \neq e \text{)}]}{\text{handle}_e(t, u) \rightarrow t} \quad \text{(h2)}$$

$$\frac{t \rightarrow u}{\text{handle}_e(t, p) \rightarrow \text{handle}_e(u, p)} \quad \text{(c1)}$$

$$\frac{t \rightarrow u}{vt \rightarrow vu} \quad \text{(c2)}$$

Operational Semantics

$$\frac{}{(\lambda x.t)v \rightarrow t\{x \setminus v\}} \quad \text{(b)}$$

$$\frac{}{v \text{ raise}_e() \rightarrow \text{raise}_e()} \quad \text{(r)}$$

$$\frac{}{\text{handle}_e(\text{raise}_e(), t) \rightarrow t} \quad \text{(h1)}$$

$$\frac{[\text{isvalue}(t)] \text{ or } [\text{israise}_{e'}(t) \text{ (} e' \neq e \text{)}]}{\text{handle}_e(t, u) \rightarrow t} \quad \text{(h2)}$$

$$\frac{t \rightarrow u}{\text{handle}_e(t, p) \rightarrow \text{handle}_e(u, p)} \quad \text{(c1)}$$

$$\frac{t \rightarrow u}{vt \rightarrow vu} \quad \text{(c2)}$$

Weak reduction: we do not reduce inside abstractions

We allow *open* normal forms



Operational Semantics Example



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

$(0 \# \text{b-steps}, 0 \# \text{exceptions handled}, 0 \# \text{exceptions propagated})$





Operational Semantics Example



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

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





Operational Semantics Example

$$\begin{array}{l} (\lambda x.\text{handle}_e(xy, x)) \quad (\lambda z.\text{raise}_e()) \\ \rightarrow_b \quad \text{handle}_e((\lambda z.\text{raise}_e()) y, (\lambda z.\text{raise}_e())) \end{array}$$

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



Operational Semantics Example

$$\begin{aligned} & (\lambda x.\text{handle}_e(xy, x)) \quad (\lambda z.\text{raise}_e()) \\ \rightarrow_b & \text{handle}_e((\lambda z.\text{raise}_e()) \quad y, (\lambda z.\text{raise}_e())) \end{aligned}$$

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

Operational Semantics Example

$(\lambda x.\text{handle}_e(xy, x))$ $(\lambda z.\text{raise}_e())$
 \rightarrow_b $\text{handle}_e((\lambda z.\text{raise}_e())$ y , $(\lambda z.\text{raise}_e())$)
 \rightarrow_b $\text{handle}_e(\text{raise}_e(), \lambda z.\text{raise}_e())$

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

Operational Semantics Example

$(\lambda x.\text{handle}_e(xy, x)) (\lambda z.\text{raise}_e())$
 $\rightarrow_b \text{handle}_e((\lambda z.\text{raise}_e()) y, (\lambda z.\text{raise}_e()))$
 $\rightarrow_b \text{handle}_e(\text{raise}_e(), \lambda z.\text{raise}_e())$

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

Operational Semantics Example

$(\lambda x.\text{handle}_e(xy, x)) (\lambda z.\text{raise}_e())$
 $\rightarrow_b \text{handle}_e((\lambda z.\text{raise}_e()) y, (\lambda z.\text{raise}_e()))$
 $\rightarrow_b \text{handle}_e(\text{raise}_e(), \lambda z.\text{raise}_e())$
 $\rightarrow_{h1} \lambda z.\text{raise}_e()$

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



Encoding Arrow Types





Encoding Arrow Types



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

Encoding Arrow Types

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

Encoding Arrow Types

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

Encoding Arrow Types

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

- $!A$ is an intersection of value types

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

Encoding Arrow Types

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

- $!A$ is an intersection of value types

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

- T is the exceptions monad

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

Encoding Arrow Types

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

- $!A$ 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	<code>tt</code>	<code>::=</code>	<code>v a n</code>
Value Types		<code>::=</code>	
Multi-types		<code>::=</code>	
Liftable Types		<code>::=</code>	
Types		<code>::=</code>	

- Computations





Types



- Values and Neutral Forms

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

- Computations



Types



- Values and Neutral Forms

Tight Constants $tt ::= v \mid a \mid n$

Value Types $\sigma ::= v \mid a \mid \mathcal{M} \Rightarrow \delta$

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 $tt ::= v \mid a \mid n$

Value Types $\sigma ::= v \mid a \mid \mathcal{M} \Rightarrow \delta$

Multi-types $\mathcal{M} ::= [\sigma_i]_{i \in I}$ where I is a finite set

Liftable Types $\mu ::= v \mid a \mid \mathcal{M}$

Types $::=$

- Computations



Types



- Values and Neutral Forms

Tight Constants $tt ::= v \mid a \mid n$

Value Types $\sigma ::= v \mid a \mid \mathcal{M} \Rightarrow \delta$

Multi-types $\mathcal{M} ::= [\sigma_i]_{i \in I}$ where I is a finite set

Liftable Types $\mu ::= v \mid a \mid \mathcal{M}$

Types $\tau ::= tt \mid \mathcal{M}$

- Computations



Types



- Values and Neutral Forms

Tight Constants $tt ::= v \mid a \mid n$

Value Types $\sigma ::= v \mid a \mid \mathcal{M} \Rightarrow \delta$

Multi-types $\mathcal{M} ::= [\sigma_i]_{i \in I}$ where I is a finite set

Liftable Types $\mu ::= v \mid a \mid \mathcal{M}$

Types $\tau ::= tt \mid \mathcal{M}$

- Computations

Monadic Types $::=$



Types



- Values and Neutral Forms

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

- Computations

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



Type System





Type System



- Judgments are decorated with counters

(*b* , *h* , *p* , *s*)





Type System



- Judgments are decorated with counters

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



Type System

- Judgments are decorated with counters

$$\underbrace{\quad}_{\# \beta\text{-steps}} (\underbrace{b \quad , \quad h \quad , \quad p \quad}_{\# \text{exceptions handled}} \quad , \quad s)$$

Type System

- Judgments are decorated with counters

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

Type System

- Judgments are decorated with counters

$$\underbrace{\quad}_{\# \beta\text{-steps}} \quad \underbrace{\quad}_{\# \text{exceptions propagated}} \\ (b \quad , \quad h \quad , \quad p \quad , \quad s) \\ \underbrace{\quad}_{\# \text{exceptions handled}} \quad \underbrace{\quad}_{|\text{normal form}|}$$



Type System



- Judgments are decorated with counters

$$\underbrace{\quad}_{\# \beta\text{-steps}} \quad \underbrace{\quad}_{\# \text{exceptions propagated}} \\ (b \quad , \quad h \quad , \quad p \quad , \quad s) \\ \underbrace{\quad}_{\# \text{exceptions handled}} \quad \underbrace{\quad}_{|\text{normal form}|}$$

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





Type System



- Judgments are decorated with counters

$$\underbrace{\quad}_{\# \beta\text{-steps}} \quad \underbrace{\quad}_{\# \text{exceptions propagated}} \\ (b \quad , \quad h \quad , \quad p \quad , \quad s) \\ \underbrace{\quad}_{\# \text{exceptions handled}} \quad \underbrace{\quad}_{|\text{normal form}|}$$

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

Type System

- Judgments are decorated with counters

$$\underbrace{\quad}_{\# \beta\text{-steps}} \quad \underbrace{\quad}_{\# \text{exceptions propagated}} \\ (b \quad , \quad h \quad , \quad p \quad , \quad s) \\ \underbrace{\quad}_{\# \text{exceptions handled}} \quad \underbrace{\quad}_{|\text{normal form}|}$$

- 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



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





(Some) Typing Rules



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

(Some) Typing Rules

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

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

(Some) Typing Rules

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

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

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



Exact Measures ()



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



Exact Measures ()

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

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

$$\begin{array}{c}
 \frac{}{x : [\sigma] \vdash^{(0,0,0,0)} x : [v] \Rightarrow (\gamma \oplus \tau)} \text{(ax)} \quad \frac{}{y : [v] \vdash^{(0,0,0,0)} y : v} \text{(ax)} \\
 \frac{}{y : [v] \vdash^{(0,0,0,0)} y : v} \text{(many)} \quad \frac{}{y : [v] \vdash^{(0,0,0,0)} y : [v]} \text{(unit)} \\
 \frac{}{y : [v] \vdash^{(0,0,0,0)} y : \star \oplus [v]} \text{(app)} \\
 \hline
 x : [\sigma], y : [v] \vdash^{(\mathbf{1},0,0,\mathbf{0})} xy : \gamma \oplus \tau
 \end{array}$$



(Some) Typing Rules Persistent



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

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

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



Exact Measures ()



Exact Measures ()

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

Exact Measures ()


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

$$\underbrace{|xy| = 1}_{xy} \not\rightarrow$$


Exact Measures (Correct)

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

$$\underbrace{|xy| = 1}_{xy} \not\rightarrow$$




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 $\exists u \in \text{no}$, s.t. $t \rightarrow (b, h, p) u$, and $|u| = s$.

Completeness



Validity of the Model



Soundness

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

Completeness

Validity of the Model

Soundness

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

Completeness

If $t \rightarrow (b, h, p) u$,

Validity of the Model

Soundness

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

Completeness

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



Typing Example



Let us consider the term exemplifying the operational semantics:

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





Typing Example



- Let us consider the term exemplifying the operational semantics:

$$(\lambda x. \text{handle}_e(xy, x)) (\lambda z. \text{raise}_e()) \rightarrow_{(2, 1, 0)} \underbrace{|\lambda z. \text{raise}_e()|}_{\lambda z. \text{raise}_e()} = \boxed{0}$$



Typing Example

- Let us consider the term exemplifying the operational semantics:

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

- We can build the following **tight** derivation:

$$\frac{}{y : [] \vdash_{(2,1,0,0)} (\lambda x. \text{handle}_e(xy, x)) (\lambda z. \text{raise}_{e'}()) : \star \oplus a} \quad \begin{array}{c} \phi \quad \psi \\ \text{(app)} \end{array}$$



Conclusion



Summary

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



Conclusion



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, $\lambda!$ -calculus, EE-calculus, ...



The End

