Elaborating evaluation-order polymorphism

Joshua Dunfield

University of British Columbia

ICFP 2015
(prologue)

▶ ICFP in Canada for the first time since 2008
ICFP in Canada for the first time since 2008

... but this is the only paper from Canada that got in
Evaluation order

Begin at the beginning.

- the Algol-60 committee meets in Paris
Evaluation order

Begin at the beginning.

- the Algol-60 committee meets in Paris
- invents call-by-name;
  decides to support it **and** call-by-value
Evaluation order

Begin at the beginning.

- the Algol-60 committee meets in Paris
- invents call-by-name;
  decides to support it **and** call-by-value
- everyone goes home
Evaluation order

Begin at the beginning.

- the Algol-60 committee meets in Paris
- invents call-by-name; decides to support it **and** call-by-value
- everyone goes home
- Peter Naur makes `cbn` the default (one of “a few matters of detail”)
Naur “had absorbed the Holy Ghost after the Paris meeting... there was nothing one could do... it was to be swallowed for the sake of loyalty.” (F.L. Bauer)
And ever since, there has been moping
Actually, I’ll take your front yard too

- Cbn languages borrowing cbv
  - Haskell strictness annotations

- Cbv languages borrowing cbn
  - SML/NJ lazy keyword
  - Scala =>
PROGRAMMERS CHOOSE A LANGUAGE BASED, IN PART, ON WHICH EVALUATION ORDER THEY USUALLY NEED
Programmers choose a language based, in part, on which evaluation order they usually need
and then tend to solve problems suited to that language’s evaluation order
Programmers choose a language based, in part, on which evaluation order they usually need and then tend to solve problems suited to that language’s evaluation order confirming their Objective Opinion about which evaluation order is “better”. 
Programmers choose a language based, in part, on which evaluation order they usually need and then tend to solve problems suited to that language’s evaluation order confirming their Objective Opinion about which evaluation order is “better”.

Is the question “which evaluation order?” that different from...
Programmers choose a language based, in part, on which evaluation order they usually need and then tend to solve problems suited to that language’s evaluation order confirming their Objective Opinion about which evaluation order is “better”.

Is the question “which evaluation order?” that different from…

“Should a language have floating-point numbers… or integers?!?”
A different question

Can we design languages that are unbiased about evaluation order?
A different question

Can we design languages that are unbiased about evaluation order?

- Types keep out bad programs
A different question

Can we design languages that are unbiased about evaluation order?

- Types keep out bad programs
- Types are also about more programs!
A different question

Can we design languages that are **unbiased** about evaluation order?

- Types keep out bad programs
- Types are also about **more** programs!
- Duplicate the **types**:
  - two \(\to\) forms: \(\rightarrow^V\), \(\rightarrow^N\)
  - two \(\ast\) forms: \(\ast^V\), \(\ast^N\)
  - evaluation-order polymorphism \(\exists_{a.\tau}\):
    write once, instantiate \(a = V\) and/or \(a = N\)
Road map

Source language (e)

Impartial type system

\[ V \rightarrow *V + V \mu V \]
\[ N \rightarrow *N + N \mu N \]
\[ \forall D \]

Impartial type system

\[ e \leftrightarrow \tau \rightarrow e \leftrightarrow S \]
encode types

Economical type system

\[ \rightarrow * + \mu \]
\[ \forall D \]

Economical type system

\[ er(e) : S \rightarrow M : A \]
elaborate

Target language (M)

Cbv type system

\[ \rightarrow * + \mu \]
\[ U \text{ (thunk)} \]
\[ \forall \]

erase types

\[ D : \text{“for all” over evaluation order} \]
Road map

Source language (e)

Impartial type system

\[ V \rightarrow \ast V + V \mu V \]
\[ N \rightarrow \ast N + N \mu N \]
\[ \forall \ D \]

\[ e \leftarrow \tau \]

encode types

\[ e \leftarrow S \]

Economical type system

\[ \rightarrow \ast + \mu \]

Target language (M)

Cbv type system

\[ \rightarrow \ast + \mu \]

\[ U \ (\text{thunk}) \]
\[ \forall \]

erase types

\[ \text{er}(e) : S \]

elaborate

\[ M : \Lambda \]

- \[ \forall \]: “for all” over evaluation order
- \[ \forall V \rightarrow S \] and \[ \forall N \rightarrow S \]: suspension type
Road map

Source language (e)

Impartial type system

\[ \begin{align*}
V & \rightarrow * V + V \mu V \\
N & \rightarrow * N + N \mu N \\
\forall & \quad \Delta
\end{align*} \]

Economical type system

\[ \begin{align*}
\rightarrow & \quad * + \mu \\
V & \triangleright N & \triangleright \Delta
\end{align*} \]

\[ e \leadsto \tau \rightarrow e \leadsto S \]

encode types

Target language (M)

Cbv type system

\[ \rightarrow * + \mu \]

\[ U \quad \text{(thunk)} \]

\[ \forall \]

\[ \begin{align*}
er(e) : S & \leftarrow M : A
\end{align*} \]

elaborate

- \( \Delta \): “for all” over evaluation order
- \( V \triangleright S \) and \( N \triangleright S \): suspension type
- \( U A \): thUnk (unit \( \rightarrow A \))
Source expression syntax

Program variables $\chi$

Source expressions $e ::= ()$

| $\chi$
| $\lambda x. e$
| $e_1 @ e_2$
| $\Lambda \alpha. e$
| $e[\tau]$
| $(e:\tau)$
| $(e_1, e_2)$
| $\text{proj}_k e$
| $\text{inj}_k e$
| $\text{case}(e, x_1.e_1, x_2.e_2)$

Expression forms work for both $\rightarrow^V$ and $\rightarrow^N$
Source expression syntax

Program variables \( x \)

Source expressions \( e \) ::=

- ( )
- \( x \)
- \( \lambda x. e \)
- \( e_1 \odot e_2 \)
- \( \Lambda \alpha. e \)
- \( e [\tau] \)
- \( (e : \tau) \)
- \( (e_1, e_2) \)
- \( \text{proj}_k e \)
- \( \text{inj}_k e \)
- \( \text{case}(e, x_1.e_1, x_2.e_2) \)

- Expression forms work for both \( \rightarrow^V \) and \( \rightarrow^N \)
- Expression forms work for both \( *^V \) and \( *^N \)
Source expression syntax

Program variables \( x \)
Source expressions \( e \) ::= ( )
  | \( x \)
  | \( \lambda x. e \)
  | \( e_1 \odot e_2 \)
  | \( \Lambda \alpha. e \)
  | \( e[\tau] \)
  | \( (e:\tau) \)
  | \( (e_1, e_2) \)
  | \( \text{proj}_k e \)
  | \( \text{inj}_k e \)
  | \( \text{case}(e, x_1.e_1, x_2.e_2) \)

- Expression forms work for both \( \rightarrow^V \) and \( \rightarrow^N \)
- Expression forms work for both \( *^V \) and \( *^N \)
- Expression forms work for both \( +^V \) and \( +^N \)
Impartial types

Evaluation order vars. \( a \)

Evaluation orders \( \epsilon ::= V \mid N \mid a \)

Type variables \( \alpha \)

Source types \( \tau ::= 1 \mid \alpha \mid \forall \alpha. \tau \mid D_a. \tau \mid \tau_1 \to \tau_2 \mid \tau_1 \ast \tau_2 \mid \tau_1 + \tau_2 \mid \mu \alpha. \tau \)
Impartial types: lists+streams

Traditional encoding of datatypes into $+$, $\mu$:

```plaintext
datatype List $\alpha =$
   Nil
| Cons of ($\alpha \ast$ List $\alpha$)
```

$\downarrow$

$\mu^1_{\beta}$. 

Generalize over evaluation orders:

type List $a$ $\alpha =$
   $\mu^1_{a}$
| $(1 + a(a \ast a \beta))$
Impartial types: lists+streams

Traditional encoding of datatypes into $+$, $\mu$:

```
datatype List $\alpha =$
    Nil of 1
| Cons of ($\alpha \ast$ List $\alpha$)
```
Impartial types: lists+streams

Traditional encoding of datatypes into $+$, $\mu$:

```
datatype List $\alpha =$ Nil of 1 |
| Cons of ($\alpha \ast \text{List } \alpha$)  
$\Rightarrow$ Nil of 1 + Cons of ($\alpha \ast \text{List } \alpha$)
```
Impartial types: lists+streams

Traditional encoding of datatypes into +, μ:

\[
\text{datatype List } \alpha = \begin{cases} 
\text{Nil of } 1 \\
\mid \text{Cons of } (\alpha \ast \text{List } \alpha)
\end{cases} \Rightarrow 1 + (\alpha \ast \text{List } \alpha)
\]
Impartial types: lists + streams

Traditional encoding of datatypes into $+$, $\mu$:

```
datatype List $\alpha =$
    Nil of 1  ⇒  1
| Cons of ($\alpha * List \alpha$) + ($\alpha * List \alpha$)

⇓

$\mu \beta. 1 + (\alpha * \beta)$
```
Impartial types: lists+streams

Traditional encoding of datatypes into +, μ:

\[
\text{datatype List } \alpha = \begin{cases} 
\text{Nil} & \Rightarrow 1 \\
| \text{Cons of } (\alpha \ast \text{List } \alpha) & + (\alpha \ast \text{List } \alpha) 
\end{cases}
\]

\[
\Downarrow \\
\mu \beta. 1 + (\alpha \ast \beta)
\]

Generalize over evaluation orders:

\[
\text{type List } a \alpha = \mu a \beta. (1 + a (\alpha \ast a \beta))
\]
Impartial types: lists+streams

\[
type \text{ List } \alpha \equiv \mu^{\alpha} \beta. (1 + ^{\alpha} (\alpha \times^{\alpha} \beta))
\]

List \( V \alpha \equiv \mu V \beta. (1 +^{V} (\alpha \times^{V} \beta)) \)  list of \( \alpha \)

List \( N \alpha \equiv \mu N \beta. (1 +^{N} (\alpha \times^{N} \beta)) \)  (terminable) stream of \( \alpha \)
Impartial types: lists+streams

\[
\text{type } \text{List } a \alpha = \mu_a \beta. (1 + a (\alpha \ast a \beta))
\]

\[
\text{List } V \alpha = \mu^V \beta. (1 +^V (\alpha \ast^V \beta)) \quad \text{list of } \alpha
\]

\[
\text{List } N \alpha = \mu^N \beta. (1 +^N (\alpha \ast^N \beta)) \quad \text{(terminable) stream of } \alpha
\]

Write \textit{map} once, for both lists and streams:

\[
\text{map} : \exists a . \forall \alpha. (\alpha \rightarrow^V \beta) \rightarrow^V (\text{List } a \alpha) \rightarrow^V (\text{List } a \beta)
\]

\[
= \forall \alpha. \text{fix map}. \lambda f. \lambda xs.
\quad \text{case}(xs, \chi_1.\text{inj}_1(),
\quad \chi_2.\text{inj}_2(f \odot (\text{proj}_1 \chi_2), \text{map} \odot f \odot (\text{proj}_2 \chi_2)))
\]
Impartial typing

Type safety holds for the simply-typed $\lambda$-calculus, regardless of evaluation order:

\[
\begin{align*}
\Gamma, x : A & \vdash M : B \\
\Gamma & \vdash \lambda x. M : A \rightarrow B \\
\Gamma & \vdash M : A \rightarrow B \\
\Gamma & \vdash N : A \\
\Gamma & \vdash MN : B
\end{align*}
\]

\[\rightarrow\text{-INTRO}\]

\[\rightarrow\text{-ELIM}\]
Impartial typing

Type safety holds for the simply-typed $\lambda$-calculus, regardless of evaluation order:

$$
\Gamma, x : A \vdash M : B \\
\frac{}{\Gamma \vdash \lambda x. M : A \rightarrow B} \rightarrow\text{-INTRO}
$$

$$
\Gamma \vdash \lambda x. M : A \rightarrow B \\
\frac{}{\Gamma, x : A \vdash M : B} \rightarrow\text{-INTRO} \quad \epsilon
$$

$$
\Gamma \vdash M : A \quad \Gamma \vdash N : A \\
\frac{}{\Gamma \vdash MN : B} \rightarrow\text{-ELIM}
$$
Impartial typing

Type safety holds for the simply-typed \( \lambda \)-calculus, regardless of evaluation order:

\[
\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x. M : A \rightarrow B} \quad \rightarrow\text{-INTRO}
\]

\[
\frac{\Gamma \vdash M : A \rightarrow B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B} \quad \rightarrow\text{-ELIM}
\]

Evaluation order \( \epsilon \) is implicit and depends on typing: typing rules can’t be syntax-directed.

How can we avoid choosing \( \epsilon \) at random?
Impartial typing

Type safety holds for the simply-typed λ-calculus, regardless of evaluation order:

\[
\Gamma, x : A \vdash M : B \\
\frac{}{\Gamma \vdash \lambda x. M : A \rightarrow B} \rightarrow\text{-INTRO}
\]

\[
\Gamma \vdash M : A \rightarrow B \quad \Gamma \vdash N : A \\
\frac{}{\Gamma \vdash MN : B} \rightarrow\text{-ELIM}
\]

Evaluation order \(\epsilon\) is implicit and depends on typing: typing rules can’t be syntax-directed.

How can we avoid choosing \(\epsilon\) at random?

- Bidirectional typing
Bidirectional typing: why?

Some past answers:

- to handle features beyond Damas–Milner (Pierce & Turner ’00; Dunfield & Pfenning ’04; Dunfield & Krishnaswami ’13; …)
- for better (earlier) type error messages

Here:

- to make typing more predictable.
Bidirectional typing in one slide

- **Organize** the flow of information from type annotations:
  - Given $\gamma$, $e$, and a known type $\tau$, **check** $e$:
    \[
    \gamma \vdash_I e \Leftarrow \tau
    \]
  - Given $\gamma$ and $e$, **synthesize** a type for $e$:
    \[
    \gamma \vdash_I e \Rightarrow \tau
    \]
  - The type $\tau$ in the checking judgment $e \Leftarrow \tau$ is a **goal**.
Impartial bidirectional typing

Frank Pfenning’s recipe: intro rules check, elim rules synthesize.

\[
\gamma, (x \Rightarrow \tau_1) \vdash_1 e \equiv \tau_2 \quad \text{I→Intro}
\]

\[
\gamma \vdash_1 (\lambda x. e) \equiv \tau_1 \overset{\epsilon}{\rightarrow} \tau_2 \quad \text{I→Intro}
\]

\[
\gamma \vdash_1 e_1 \Rightarrow e_2 \overset{\epsilon}{\rightarrow} \tau_2 \quad \gamma \vdash_1 e_2 \equiv \tau_1 \quad \text{I→Elim}
\]

\[
\gamma \vdash_1 (e_1 \oplus e_2) \Rightarrow \tau_2
\]

- **I→Intro**: The type \( \tau_1 \overset{\epsilon}{\rightarrow} \tau_2 \) must flow from an annotation.
- **I→Elim**: The type \( \tau_1 \overset{\epsilon}{\rightarrow} \tau_2 \) must flow from an annotation, perhaps via \( \gamma \).
Impartial bidirectional typing

Frank Pfenning’s recipe:
intro rules check, elim rules synthesize.

\[
\begin{align*}
\frac{\gamma, (x \Rightarrow \tau_1) \vdash_1 e \iff \tau_2}{\gamma \vdash_1 (\lambda x. e) \iff \tau_1 \xrightarrow{\epsilon} \tau_2} \quad & \text{I→Intro} \\
\frac{\gamma \vdash_1 e_1 \Rightarrow \tau_1 \xrightarrow{\epsilon} \tau_2 \quad \gamma \vdash_1 e_2 \iff \tau_1}{\gamma \vdash_1 (e_1 \odot e_2) \Rightarrow \tau_2} \quad & \text{I→Elim}
\end{align*}
\]

- **I→Intro**: The type \(\tau_1 \xrightarrow{\epsilon} \tau_2\) must flow from an annotation.
- **I→Elim**: The type \(\tau_1 \xrightarrow{\epsilon} \tau_2\) must flow from an annotation, perhaps via \(\gamma\).
Impartial bidirectional typing

Intro rules check, elim rules synthesize.
Handle $\Delta$ like $\forall$:

$\gamma, a \text{ evalorder} \vdash_1 e \iff \tau$

\[
\frac{\gamma \vdash_1 e \iff \Delta a. \tau}{\gamma \vdash_1 e \iff \Delta a. \tau} \quad \text{(ID Intro)}
\]

\[
\frac{\gamma \vdash_1 e \Rightarrow \Delta a. \tau \quad \gamma \vdash \epsilon \text{ evalorder}}{\gamma \vdash_1 e \Rightarrow [\epsilon / a] \tau} \quad \text{(ID Elim)}
\]
Impartial bidirectional typing

Intro rules check, elim rules synthesize. Handle $\mathcal{D}$ like $\forall$:

$$
\begin{align*}
\gamma, a \text{ evalorder} & \vdash_I e \quad \iff \quad \tau \\
\gamma & \vdash_I e \quad \iff \quad \mathcal{D}a. \tau & \text{ IDIntro} \\
\gamma & \vdash_I e \quad \Rightarrow \quad \mathcal{D}a. \tau \\
\gamma & \vdash_\epsilon \quad \text{evalorder} \\
\gamma & \vdash_I e \quad \Rightarrow \quad [\epsilon / a]\tau & \text{ IDElim}
\end{align*}
$$

- Back to guessing? $\epsilon$ in IDElim
- Can force an eval. order by annotating $e$
- In practice, probably want the ability to set a default (lexically-scoped?) for IDElim.
Road map

Source language (e)

Impartial type system

\[ \begin{align*}
    V & \rightarrow * V + V \mu V \\
    N & \rightarrow * N + N \mu N \\
    \forall & \text{ } D
\end{align*} \]

Economical type system

\[ \begin{align*}
    \rightarrow & \rightarrow * + \mu \\
    \forall & \text{ } D
\end{align*} \]

\[ e \leftrightarrow \tau \rightarrow e \leftrightarrow S \]
encode types

Target language (M)

Cbv type system

\[ \rightarrow * + \mu \]

\[ \forall \text{ } D \]
erase types

\[ \text{er}(e) : S \leftarrow M : A \]

elaborate

- \( \forall D \): “for all” over evaluation order
- \( \forall V \rightarrow S \text{ and } N \rightarrow S \): suspension type
- \( \forall U A : \text{thUnk} (\text{unit} \rightarrow A) \)
Road map

Source language \((e)\)

- **Impartial type system**
  
  \[
  V \rightarrow * V + V \mu V \\
  N \rightarrow * N + N \mu N \\
  \forall \Delta
  \]

- **Economical type system**
  
  \[
  \rightarrow * + \mu \\
  \forall \Delta
  \]

  - **Encode types**
    
    \(e \leftrightarrow \tau\)

  - **Source language**
    
    \(e \leftrightarrow S\)

  - **Target language**
    
    \(er(e) : S \rightarrow M : A\)

  - **Target language**
    
    \(M : A\)

- **“for all” over evaluation order**
  
  \(\forall \Delta\)

- **V▷S and N▷S: suspension type**

- **U A: thUnk (unit \(\rightarrow A)\)**

- **Erasure types**
  
  \(\forall \Delta\)
Road map

Source language (e)

Impartial type system

经济社会 language (e)

Economical type system

Target language (M)

Cbv type system

∀ D: “for all” over evaluation order

V ▷ S and N ▷ S: suspension type

U A: thUnk (unit → A)
Economical types = by-value + suspensions

Factor out the differences between V, N connectives:

Economical types  \( S \ ::= \ 1 \mid \alpha \mid \forall \alpha. \ S \mid \exists a. \ S \mid \epsilon \triangleright S \mid S_1 \rightarrow S_2 \mid S_1 \ast S_2 \mid S_1 + S_2 \mid \mu \alpha. \ S \)

\( \triangleright S \) “N suspend S”  \( \mapsto \ 1 \rightarrow S \)

\( \triangleright S \) “V suspend S”  \( \mapsto \ S \)
Economical types = by-value + suspensions

Factor out the differences between V, N connectives:

Economical types \( S := 1 \mid \alpha \mid \forall \alpha. S \mid D\alpha. S \mid \epsilon \triangleright S \mid S_1 \rightarrow S_2 \mid S_1 \ast S_2 \mid S_1 + S_2 \mid \mu \alpha. S \)

\( N\triangleright S \) “N suspend S” \( \leftrightarrow 1 \rightarrow S \)
\( \forall\triangleright S \) “V suspend S” \( \leftrightarrow S \)

Encode impartial types as economical types:

\[
\begin{align*}
[1] & = 1 \\
[\tau_1 \rightarrow^e \tau_2] & = (\epsilon\triangleright [\tau_1]) \rightarrow [\tau_2] \\
[\tau_1 \ast^e \tau_2] & = \epsilon\triangleright ([\tau_1] + [\tau_2]) \\
[\alpha] & = \alpha
\end{align*}
\]
\[
\begin{align*}
[\forall \alpha. \tau] & = \forall \alpha. [\tau] \\
[\mu^e \alpha. \tau] & = \mu \alpha. \epsilon\triangleright [\tau] \\
[\epsilon \triangleright S] & = \epsilon\triangleright S
\end{align*}
\]

\( D\alpha. \tau \)
Economical types: lists + streams

\[
\text{type List } a \ \alpha = \mu\beta. a \triangleright (1 + (\alpha \ast \beta))
\]

\[
\text{List } V \ \alpha = \mu\beta. V \triangleright (1 + (\alpha \ast \beta)) \quad \text{list of } \alpha
\]

\[
\text{List } N \ \alpha = \mu\beta. N \triangleright (1 + (\alpha \ast \beta)) \quad \text{(terminable) stream of } \alpha
\]

Write \textit{map} once, for both lists and streams:

\[
\text{map} : \Delta a. \forall \alpha. (\alpha \to \beta) \to (\text{List } a \ \alpha) \to (\text{List } a \ \beta)
\]

\[
= \Lambda \alpha. \text{fix } \text{map} \cdot \lambda f. \lambda xs. \\
\quad \text{case}(xs, x_1.\text{inj}_1(), \\
\quad \quad x_2.\text{inj}_2(f \circ (\text{proj}_1 x_2), \ \text{map} \circ f \circ (\text{proj}_2 x_2)))
\]

(body of \textit{map} unchanged from the impartial system)
Road map

Source language \((e)\)

- Impartial type system
  \[
  \begin{align*}
  V & \rightarrow \ast \lor \lor \lor \mu V \\
  N & \rightarrow \ast N \lor N \lor \mu N \\
  \forall & \Delta
  \end{align*}
  \]

- Encode types
  \[
  e \leftrightarrow \tau \quad \rightarrow e \leftrightarrow S
  \]

Economical type system

\[
\begin{align*}
  \rightarrow & \ast + \mu \\
  \forall & \Delta
\end{align*}
\]

Target language \((M)\)

- Cbv type system
  \[
  \rightarrow \ast + \mu
  \]

- Erase types
  \[
  \forall
  \]

- Elaborate
  \[
  \text{er}(e) : S \quad \rightarrow M : A
  \]

- \(\Delta\): “for all” over evaluation order
- \(\lor S\) and \(\lor S\): suspension type
- \(U A\): thUnk (unit \(\rightarrow A\))
Road map

Source language (e)

Impartial type system
\[ \forall \mathcal{D} \]
\[ V \rightarrow * V + V \mu V \]
\[ N \rightarrow * N + N \mu N \]

Economical type system
\[ \rightarrow * + \mu \]
\[ V \uparrow \]
\[ N \uparrow \]
\[ \forall \mathcal{D} \]

\[ e \leftrightarrow \tau \rightarrow e \leftrightarrow S \]
encode types

Target language (M)

Cbv type system
\[ \rightarrow * + \mu \]
\[ \forall \]
\[ U \text{ (thunk)} \]
erase types

\[ \forall \]

\[ er(e) : S \rightarrow M : A \]
elaborate

- \( \mathcal{D} \): “for all” over evaluation order
- \( \triangleright V S \) and \( \triangleright N S \): suspension type
- \( U A : \text{thUnk (unit } \rightarrow A) \)
Road map

Source language \((e)\)

**Impartial type system**

\[
\begin{align*}
V & \rightarrow * V + \mu V \\
N & \rightarrow * N + \mu N \\
\forall \ D
\end{align*}
\]

\[
\begin{array}{c}
e \leftrightarrow \tau \\
\text{encode types}
\end{array}
\]

\[
\begin{array}{c}
e \leftrightarrow S \\
\text{er}(e) : S \\
\text{elaborate}
\end{array}
\]

**Economical type system**

\[
\begin{align*}
\rightarrow & \rightarrow * + \mu \\
\forall \ D
\end{align*}
\]

Target language \((M)\)

**Cbv type system**

\[
\begin{align*}
\rightarrow & \rightarrow * + \mu \\
\forall
\end{align*}
\]

**erase types**

\[
\text{U (thunk)}
\]

\[
\begin{array}{c}
M : A
\end{array}
\]

- \(\forall \ D\): “for all” over evaluation order
- \(\forall S \rightarrow S\) and \(\forall S\): suspension type
- \(\text{U A}\): thUnk \((\text{unit} \rightarrow A)\)
Elaboration

For the metatheory, elaboration not bidirectional. For every economical typing with

\[ e \iff S \quad \text{or} \quad e \implies S \]

we can erase types from \( e \) and elaborate to \( M \)

\[ \text{er}(e) : S \hookrightarrow M \]
Elaboration: rules for ▶

\[ \Gamma \vdash e : S \leftrightarrow M \]

\[ \Gamma \vdash e : V \uparrow S \leftrightarrow M \quad \Gamma \vdash e : N \uparrow S \leftrightarrow \text{thunk} M \]

\[ \Gamma \vdash e : V \uparrow S \leftrightarrow M \]

\[ \Gamma \vdash e : S \leftrightarrow M \quad \text{elab} \uparrow \text{Intro} \]

\[ \Gamma \vdash e : N \uparrow S \leftrightarrow M \]

\[ \Gamma \vdash e : S \leftrightarrow \text{force} M \quad \text{elab} \uparrow \text{Elim}_N \]

\[ \text{▶ inj}_1 () : N \uparrow (1 + \cdots) \leftrightarrow \text{thunk} \ (\text{inj}_1 ()) \]

\[ \text{▶ Given } y : N \uparrow (1 + \cdots), \]

\[ y : (1 + \cdots) \leftrightarrow \text{force } y \]
Elaboration: rules for $\to$ 

\[
\begin{align*}
\Gamma \vdash e : S & \leftrightarrow M \\
\Gamma \vdash e : V \to S & \leftrightarrow M \\
\Gamma \vdash e : N \to S & \leftrightarrow \text{thunk } M \\
\Gamma \vdash e : V \to S & \leftrightarrow M \\
\Gamma \vdash e : S & \leftrightarrow M \\
\Gamma \vdash e : N \to S & \leftrightarrow M \\
\Gamma \vdash e : S & \leftrightarrow \text{force } M
\end{align*}
\]

- $\to$ $\text{intro}$
- $\to$ $\text{elim}_V$
- $\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$

$\to$ $\text{intro}$

$\to$ $\text{elim}_V$

$\to$ $\text{elim}_N$
Elaboration: rules for $\Delta$

\[
\begin{align*}
\Gamma \vdash e : [V/a]S &\rightarrow M_1 & \Gamma \vdash e : [N/a]S &\rightarrow M_2 \\
\Gamma \vdash e : (\Delta a . S) &\rightarrow (M_1, M_2) \\
\end{align*}
\]

$elab\Delta Intro$

\[
\begin{align*}
\Gamma \vdash e : (\Delta a . S) &\rightarrow M \\
\Gamma \vdash e : [V/a]S &\rightarrow proj_1 M & \Gamma \vdash e : [N/a]S &\rightarrow proj_2 M \\
\end{align*}
\]

$elab\Delta Elim$

$\bullet$ $map : \Delta a . \cdots \rightarrow (map_V, map_N)$

$\bullet$ Given $xs : \text{List } N \beta$,

\[
map \circ f \circ xs \rightarrow (proj_2 map) f xhs
\]
Elaboration: rules for $\mathcal{D}$

\[
\Gamma \vdash e : [V/a]S \rightarrow M_1 \quad \Gamma \vdash e : [N/a]S \rightarrow M_2
\]

\[
\Gamma \vdash e : (\mathcal{D}a. S) \rightarrow (M_1, M_2)
\]

\[
\Gamma \vdash e : (\mathcal{D}a. S) \rightarrow M
\]

\[
\Gamma \vdash e : [V/a]S \rightarrow \text{proj}_1 M \quad \Gamma \vdash e : [N/a]S \rightarrow \text{proj}_2 M
\]

- $map : \mathcal{D}a. \cdots \rightarrow (map_V, map_N)$

- Given $xs : \text{List } N \beta$, 

\[
map \odot f \odot xs \rightarrow (\text{proj}_2 map) f \, xs
\]
Elaboration: rules for $\Delta$

\[
\begin{align*}
\Gamma \vdash e : [V/a]S & \rightarrow M_1 & \Gamma \vdash e : [N/a]S & \rightarrow M_2 \\
\hline
\Gamma \vdash e : (\Delta a. S) & \rightarrow (M_1, M_2) \\
\Gamma \vdash e : (\Delta a. S) & \rightarrow M  \\
\Gamma \vdash e : [V/a]S & \rightarrow \text{proj}_1 M & \Gamma \vdash e : [N/a]S & \rightarrow \text{proj}_2 M
\end{align*}
\]

- $\map : \Delta a. \cdots \rightarrow (\map_V, \map_N)$

- Given $xs : \text{List } N \beta$,

\[
\map \odot f \odot xs \rightarrow (\text{proj}_2 \map) f \, xs
\]

Only instantiating $\Delta a$ with $N$ will succeed.
Metatheory

Source language \((e)\)

Impartial type system

\[
\frac{}{V \rightarrow \ast \ast V + \mu \mu V}
\]

\[
\frac{}{N \rightarrow \ast N + N \mu N}
\]

\[
\forall \Gamma
\]

Impartial type system

\[
\frac{}{V \rightarrow \ast \ast V + \mu \mu V}
\]

\[
\frac{}{N \rightarrow \ast N + N \mu N}
\]

\[
\forall \Gamma
\]

Economical type system

\[
\frac{}{\rightarrow \ast + \mu}
\]

\[
\forall \Gamma
\]

Economical type system

\[
\frac{}{V \triangleright N \triangleright}
\]

\[
\forall \Gamma
\]

\[
\frac{}{e \triangleleft \tau}
\]

encode types

\[
\frac{}{e \triangleleft S}
\]

encode types

\[
\frac{}{er(e) : S}
\]

erase types

\[
\frac{}{\frac{}{M : A}}
\]

elaborate

Target language \((M)\)

Cbv type system

\[
\frac{}{\rightarrow \ast + \mu}
\]

\[
\forall
\]

\[
\frac{}{U (thunk)}
\]
**Metatheory: consistency**

**Source language** (e)

- Impartial type system:
  \[ V \rightarrow \ast V + V \mu V \]
  \[ N \rightarrow \ast N + N \mu N \]
  \[ \forall \Delta \]

- Encode types:
  \( e \triangleleft \tau \rightarrow e \triangleleft S \)

**Economical type system**

- Erase types:
  \[ V \triangleright N \triangleright \]
  \[ \forall \Delta \]

- Encode types:
  \( e \triangleleft S \)

**Target language** (M)

- Cbv type system:
  \[ \rightarrow \ast + \mu \]
  \[ U \text{ (thunk)} \]
  \[ \forall \]

- Elaborate:
  \[ er(e) : S \rightarrow M : A \]

- Standard cbv evaluation:
  \( (\geq 0 \text{ steps}) \)

- W : A

If \( er(e) : S \leftrightarrow M \) and \( M \rightarrow^* W \)
**Metatheory: consistency**

**Source language** (e)

**Impartial type system**
\[
V \rightarrow \ast \rightarrow V + \mu V \\
N \rightarrow \ast N + \mu N \\
\forall \Delta
\]

**Economical type system**
\[
\rightarrow \ast + \mu \\
V' \rightarrow N' \\
\forall \Delta
\]

Erase types

\[e \iff \tau \rightarrow e \iff S\]

Encode types

\[er(e) : S \rightarrow M : A\]

Target language (M)

**Cbv type system**
\[
\rightarrow \ast + \mu \\
U \text{ (thunk)} \\
\forall
\]

Economical type system

\[\forall \Delta\]

Encode types

\[cbv + cbn\]

Evaluation

\[\geq 0 \text{ steps}\]

\[M \rightarrow \ast \rightarrow W : A\]

Standard cbv evaluation

\[\geq 0 \text{ steps}\]

\[e' : S \rightarrow W : A\]

If \(er(e) : S \rightarrow M\) and \(M \rightarrow \ast \rightarrow W\) then \(er(e) \rightarrow \ast \rightarrow e'\) and \(e' : S \rightarrow W\).
Metatheory: $N$-freeness

Theorem.
If an impartial typing judgment with $\tau$ is $N$-free, then the corresponding economical judgment with $[\tau]$ is $N$-free.

Theorem.
If an economical typing judgment with $e$ is $N$-free then $er(e)$ elaborates to $M$, and $M$ is $N$-free.

Theorem (Multi-step consistency).

\ldots

Moreover, if $M$ is $N$-free then we can derive $e \rightsquigarrow^* e'$ without by-name reductions.

- A judgment with $\Delta$ is not $N$-free (subformula property).
History

- Church: normal-order (leftmost-outermost) reduction
- Bernays (1936), reviewing Church & Rosser (1936): require arguments in normal form (not head-normal form)
- Naur et al. (1960): Algol-60 with cbn/cbv on per-argument basis (resentful committee members: HOPL)
Immediate ancestors

- Chen et al., ICFP 2011: elaboration from intersection types (we just didn’t call them that), for incremental computation

- ICFP 2012: elaboration from intersection types; like $\mathcal{D}$, but more general and less predictable: here we only “intersect” $V$- and $N$-instantiations
Immediate ancestors

- Chen et al., ICFP 2011: elaboration from intersection types (we just didn’t call them that), for incremental computation
- ICFP 2012: elaboration from intersection types; like $\Delta$, but more general and less predictable: here we only “intersect” V- and N-instantiations
What’s next?

- an implementation?
- let? lexically-scoped default ε?
- call-by-need

Could these techniques resolve other “fundamental” differences between PLs?
What’s next?

- an implementation?
- let? lexically-scoped default ε?
- call-by-need

Could these techniques resolve other “fundamental” differences between PLs?

One big happy family (united against the OOP enemy?)
Conclusion:

Evaluation-order peace may be on its way!

Paper and proofs:  arxiv.org/abs/1504.07680

Special thanks to: the cats of Metro Vancouver and Cambridge, England

Shameless advertisement:
I’m looking for a research/teaching job in Canada.
Conclusion:
Evaluation-order peace may be on its way!

Paper and proofs: arxiv.org/abs/1504.07680

Special thanks to: the cats of Metro Vancouver and Cambridge, England

Shameless advertisement:
I’m looking for a research/teaching job in Canada.
Conclusion:
Evaluation-order peace may be on its way!

Paper and proofs:  arxiv.org/abs/1504.07680

Special thanks to: the cats of Metro Vancouver and Cambridge, England

Shameless advertisement:
I’m looking for a research/teaching job in Canada.
Target for elaboration

Standard cbv, plus thunks (type $\texttt{U B}$)

\[
\begin{align*}
\Gamma \vdash_T M : B & \quad \text{\texttt{TUIntro}} \\
\Gamma \vdash_T \text{thunk } M : \texttt{U B} & \\
\Gamma \vdash_T \text{force } M_1 : B & \quad \text{\texttt{TUElim}}
\end{align*}
\]

- thunk type $\texttt{U B}$ could be $1 \rightarrow B$
- distinguished only for the metatheory
Source operational semantics

- Define by-value and by-name reductions:

\[(\lambda x. e_1) @ v_2 \leadsto_{RV} [v_2/x]e_1\]
\[(\lambda x. e_1) @ e_2 \leadsto_{RN} [e_2/x]e_1\]

- Allow reductions in the appropriate eval. contexts:

\[C_V[e] \leadsto_{RV} C_V[e']\]
\[C_N[e] \leadsto_{RN} C_N[e']\]

Example:
\[(\lambda x. e) @ []\] is a \(C_V\) but not a \(C_N\)
The $\rightsquigarrow$ relation is “too big”, because it’s not connected to typing.

“(λx. e) @ [] is a $C_V$ but not a $C_N$”

In fact, $\rightsquigarrow$ is much too big in the paper, which defines $C_N$ incorrectly.

But the metatheory doesn’t care.

Guess I’m even more cbv-biased than I thought!