A Contextual Account of Staged Computations

Brigitte Pientka Matthias Puech

McGill University, Canada

Groupe de Travail Théorie des Types et Réalisabilité
PPS, Univ. Paris Diderot

October 29, 2014

A Contextual Account of Staged Computations

Brigitte Pientka Matthias Puech

McGill University, Canada

Groupe de Travail Théorie des Types et Réalisabilité
PPS, Univ. Paris Diderot

October 29, 2014

(WIP)

Those red apples

⟨Those red apples ⟩ is a noun phrase.

For any noun n,
$\langle$Those red $\sim$(pluraled n)$\rangle$ is a noun phrase.

$\langle$For any noun n,
$\langle$Those red $\sim$(pluraled n)$\rangle$ is a noun phrase.$\rangle$
is valid a grammar rule.

For some part of speech P, $\langle$For any $\sim(P)$ n,
$\langle$Those red $\sim$(pluraled n)$\rangle$ is a $\sim(P)$ phrase.$\rangle$
is valid a grammar rule.

For some part of speech P, ⟨For any ∼(P) n,
⟨Those red ∼(pluraled n)⟩ is a ∼(P) phrase.⟩
is valid a grammar rule.

For some part of speech P, $\langle$For any $\sim(P)$ n, $\langle$Those red $\sim$(pluraled n)$\rangle$ is a $\sim(P)$ phrase.$\rangle$ is valid a grammar rule.

This talk is about typing such metalanguages in a *principled* way.

Motivation 1: Macros

In ML, **if** is syntactic sugar:

$$(\textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3) \triangleq (\textbf{match}\ e_1\ \textbf{with}\ \textbf{true} \rightarrow e_2 \mid \textbf{false} \rightarrow e_3)$$

Motivation 1: Macros

In ML, **if** is syntactic sugar:

$$(\textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3) \triangleq (\textbf{match}\ e_1\ \textbf{with true} \rightarrow e_2 \mid \textbf{false} \rightarrow e_3)$$

Problem

We can't define it in CBV:

let `if_ e1 e2 e3` = **match** `e1` **with true** → `e2` | **false** → `e3`

Motivation 1: Macros

In ML, **if** is syntactic sugar:

$$(\textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3) \triangleq (\textbf{match}\ e_1\ \textbf{with true} \rightarrow e_2 \mid \textbf{false} \rightarrow e_3)$$

Problem

We can't define it in CBV:

let `if_ e1 e2 e3` = **match** `e1` **with true** → `e2` | **false** → `e3` **in**
`if_` **false** (`raise` Exit) (`print_string` "Hello")

Motivation 1: Macros

In ML, **if** is syntactic sugar:

$$(\textbf{if}\ \ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3) \triangleq (\textbf{match}\ e_1\ \textbf{with true} \rightarrow e_2 \mid \textbf{false} \rightarrow e_3)$$

Problem

We can't define it in CBV:

```
let if_ e1 e2 e3 = match e1 with true → e2 | false → e3 in
if_ false (raise Exit) (print_string "Hello");;
Hello Exception: Pervasives.Exit.                  (* fail *)
```

Motivation 1: Macros

In ML, **if** is syntactic sugar:

$$(\textbf{if}\ \ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3) \triangleq$$
$$(\textbf{match}\ e_1\ \textbf{with}\ \textbf{true} \to e_2 |\ \textbf{false} \to e_3)$$

Problem

We can't define it in CBV:

```
let if_ e1 e2 e3 = match e1 with true → e2 | false → e3 in
if_ false (raise Exit) (print_string "Hello");;
Hello Exception: Pervasives.Exit.                (* fail *)
```

⇝ How to define syntactic sugar in the language?

Motivation 2: Program specialization

```
let rec pow x n =
  if n = 0 then 1
  else if n mod 2 = 0 then sq (pow x (n/2))
  else x * pow x (n-1)
```

Motivation 2: Program specialization

```
let rec pow x n =
  if n = 0 then 1
  else if n mod 2 = 0 then sq (pow x (n/2))
  else x * pow x (n−1)
```

Problem

For performance, we want to derive specialized programs:

```
let pow13 x = pow x 13              (* a closure of pow *)
```

Motivation 2: Program specialization

```
let rec pow x n =
  if n = 0 then 1
  else if n mod 2 = 0 then sq (pow x (n/2))
  else x * pow x (n−1)
```

Problem

For performance, we want to derive specialized programs:

```
let pow13 x = pow x 13                       (* a closure of pow *)
let pow13s x = x * sq (sq (x * sq (x * 1)))  (* 6x faster *)
```

Motivation 2: Program specialization

```
let rec pow x n =
  if n = 0 then 1
  else if n mod 2 = 0 then sq (pow x (n/2))
  else x * pow x (n−1)
```

Problem

For performance, we want to derive specialized programs:

```
let pow13 x = pow x 13                        (* a closure of pow *)
let pow13s x = x * sq (sq (x * sq (x * 1)))   (* 6x faster *)
```

⇝ How to specialize a function on a statically known argument?

Motivation 3: Full evaluation

Evaluation stops at λs:

```
let f = fun x → ((fun y → y + 1) x);;
val f : int → int = <fun>
```

Motivation 3: Full evaluation

Evaluation stops at λs:

```
let f = fun x → ((fun y → y + 1) x);;
val f : int → int = <fun>
```

Problem

We sometimes need to syntactically compare normal forms.

```
f = fun x → x + 1;;
Exception: Invalid_argument "equal:␣functional␣value".
```

$\rightsquigarrow$ How to evaluate under λs?

One-size-fits-all solution: Staging

A multi-staged functional programming language provides a finer control over evaluation.

One-size-fits-all solution: Staging

A multi-staged functional programming language provides a finer control over evaluation.

The method

Organizes evaluation into ordered stages (level):

- each redex belongs to a stage n,
- redex n fired only if no redex $m < n$

One-size-fits-all solution: Staging

A multi-staged functional programming language provides a finer control over evaluation.

The method

Organizes evaluation into ordered stages (level):

- each redex belongs to a stage n,
- redex n fired only if no redex $m < n$

The abstraction

Generating, grafting and running pieces of code (AST).

Multi-staged languages

Syntax:

$$e \;::=\; \ldots \mid \langle e \rangle \mid {\sim}e \mid \texttt{run}\; e$$

Operational semantics:

$${\sim}\langle v \rangle \longrightarrow v \qquad\qquad \texttt{run}\;\langle e \rangle \longrightarrow e$$

$$C \;::=\; \ldots \mid \langle \ldots {\sim}C \ldots \rangle$$

Examples

Macros

$$\textbf{let}\ \texttt{if\_}\ \texttt{e1}\ \texttt{e2}\ \texttt{e3} = \langle\ \textbf{match}\ {\sim}\texttt{e1}\ \textbf{with true} \rightarrow {\sim}\texttt{e2} \mid \textbf{false} \rightarrow {\sim}\texttt{e3}\ \rangle$$

Examples

Macros

```
let if_ e1 e2 e3 =
  ⟨ match ~e1 with true → ~e2 | false → ~e3 ⟩ ;;

let e = ⟨~(if_ ⟨false⟩ ⟨raise Exit⟩ ⟨print_string "Hello"⟩)⟩
```

Examples

Macros

let if_ e1 e2 e3 =
  ⟨ **match** ~e1 **with true** → ~e2 | **false** → ~e3 ⟩ ;;

let e = ⟨~(if_ ⟨**false**⟩ ⟨raise Exit⟩ ⟨print_string "Hello"⟩)⟩;;
val e =
  ⟨ **match false with**
    | **true** → raise Exit
    | **false** → print_string "Hello" ⟩

Examples

Macros

```
let if_ e1 e2 e3 =
  ⟨ match ~e1 with true → ~e2 | false → ~e3 ⟩ ;;

let e = ⟨~(if_ ⟨false⟩ ⟨raise Exit⟩ ⟨print_string "Hello"⟩)⟩;;
val e =
  ⟨ match false with
     | true  → raise Exit
     | false  → print_string "Hello" ⟩ ;;

run e
```

Examples

Macros

```
let if_ e1 e2 e3 =
  ⟨ match ~e1 with true → ~e2 | false → ~e3 ⟩ ;;

let e = ⟨~(if_ ⟨false⟩ ⟨raise Exit⟩ ⟨print_string "Hello"⟩)⟩;;
val e =
  ⟨ match false with
     | true  → raise Exit
     | false  → print_string "Hello" ⟩ ;;

run e ;;
Hello − : unit = ()
```

Examples

Program specialization

$$
\begin{array}{l}
\textbf{let rec}\ \texttt{pow x n} = \\
\quad \textbf{if}\ \texttt{n} = 0\ \textbf{then}\ \langle 1 \rangle \\
\quad \textbf{else if}\ \texttt{n}\ \textbf{mod}\ 2 = 0\ \textbf{then}\ \langle \texttt{square}\ \sim(\texttt{pow x (n/2)}) \rangle \\
\quad \textbf{else}\ \langle \sim\texttt{x} * \sim(\texttt{pow x (n-1)}) \rangle
\end{array}
$$

Examples

Program specialization

$$\textbf{let rec}\ \texttt{pow x n} =$$
$$\quad \textbf{if}\ \texttt{n} = 0\ \textbf{then}\ \langle 1 \rangle$$
$$\quad \textbf{else if}\ \texttt{n}\ \textbf{mod}\ 2 = 0\ \textbf{then}\ \langle \texttt{square} \sim(\texttt{pow x (n/2)}) \rangle$$
$$\quad \textbf{else}\ \langle \sim\texttt{x} * \sim(\texttt{pow x (n-1)}) \rangle\ ;;$$

$$\textbf{let}\ \texttt{pow13} = \langle\ \textbf{fun}\ \texttt{x} \to \sim(\texttt{pow}\ \langle \texttt{x} \rangle\ \texttt{13})\ \rangle$$

Examples

Program specialization

```
let rec pow x n =
  if n = 0 then ⟨1⟩
  else if n mod 2 = 0 then ⟨square ~(pow x (n/2))⟩
  else ⟨~x * ~(pow x (n−1))⟩ ;;

let pow13 = ⟨ fun x → ~(pow ⟨x⟩ 13) ⟩ ;;
val pow13 =
  ⟨ fun x  → x * square (square (x * square (x * 1)))⟩
```

Examples

Program specialization

```
let rec pow x n =
  if n = 0 then ⟨1⟩
  else if n mod 2 = 0 then ⟨square ~(pow x (n/2))⟩
  else ⟨~x * ~(pow x (n−1))⟩ ;;

let pow13 = ⟨ fun x → ~(pow ⟨x⟩ 13) ⟩ ;;
val pow13 =
  ⟨ fun x  → x * square (square (x * square (x * 1)))⟩

run pow13 2
```

Examples

Program specialization

```
let rec pow x n =
  if n = 0 then ⟨1⟩
  else if n mod 2 = 0 then ⟨square ~(pow x (n/2))⟩
  else ⟨~x * ~(pow x (n−1))⟩ ;;

let pow13 = ⟨ fun x → ~(pow ⟨x⟩ 13) ⟩ ;;
val pow13 =
  ⟨ fun x  → x * square (square (x * square (x * 1)))⟩

run pow13 2 ;;
− : int = 8192
```

Examples

Full evaluation

$$\textbf{let}\ \ \mathtt{e} = \langle \textbf{fun}\ \mathrm{x} \rightarrow {\sim}((\textbf{fun}\ \mathrm{y} \rightarrow \langle\ {\sim}\mathrm{y} + 1\ \rangle)\ \langle \mathrm{x} \rangle) \rangle$$

Examples

Full evaluation

$\textbf{let}\ \ \texttt{e} = \langle \textbf{fun}\ \mathrm{x} \rightarrow {\sim}((\textbf{fun}\ \mathrm{y} \rightarrow \langle\ {\sim}\mathrm{y} + 1\ \rangle)\ \langle \mathrm{x} \rangle)\rangle\ ;;$

$\textbf{val}\ \ \texttt{e} = \langle \textbf{fun}\ \mathrm{x}\ \ \rightarrow \mathrm{x} + 1 \rangle$

Examples

Full evaluation

$$\textbf{let}\;\; \mathtt{e} = \langle \textbf{fun}\; \mathtt{x} \rightarrow \sim((\textbf{fun}\; \mathtt{y} \rightarrow \langle\; \sim\mathtt{y} + 1\; \rangle)\; \langle \mathtt{x} \rangle)\rangle\;;;$$
$$\textbf{val}\;\; \mathtt{e} = \langle \textbf{fun}\; \mathtt{x}\;\; \rightarrow \mathtt{x} + 1\rangle$$

```
run e 42
```

Examples

Full evaluation

$\textbf{let}\ \texttt{e} = \langle \textbf{fun}\ \mathrm{x} \rightarrow \sim((\textbf{fun}\ \mathrm{y} \rightarrow \langle\ \sim\mathrm{y} + 1\ \rangle)\ \langle \mathrm{x} \rangle) \rangle\ ;;$

$\textbf{val}\ \texttt{e} = \langle \textbf{fun}\ \mathrm{x}\ \rightarrow \mathrm{x} + 1 \rangle$

$\texttt{run}\ \texttt{e}\ 42\ ;;$

$-: \texttt{int} = 43$

Examples

Full evaluation

$$\textbf{let}\ \ \mathtt{e} = \langle\textbf{fun}\ \mathrm{x} \rightarrow \sim((\textbf{fun}\ \mathrm{y} \rightarrow \langle\ \sim\mathrm{y} + 1\ \rangle)\ \langle\mathrm{x}\rangle)\rangle\ ;;$$
$$\textbf{val}\ \ \mathtt{e} = \langle\textbf{fun}\ \mathrm{x}\ \ \rightarrow \mathrm{x} + 1\rangle$$

$$\mathtt{run}\ \mathtt{e}\ 42\ ;;$$
$$-:\ \mathtt{int} = 43$$

Remark

We must now ensure:

- lexical scoping (variables used in their binding context...)
- congruence (...at their binding stage) = "staged lexical scoping"
  <u>ex</u>: $\not\vdash \langle\ \textbf{fun}\ \mathrm{x} \rightarrow \sim(\mathrm{x} + 1)\ \rangle$
- evaluation of closed code
  <u>ex</u>: $\not\vdash \langle\ \textbf{fun}\ \mathrm{x} \rightarrow \sim(\mathtt{run}\ \langle\mathrm{x}\rangle)\ \rangle$

A type system for staged computations?

$\lambda\Box$ (Davies & Pfenning, 1996)

$\Delta; \Gamma \vdash M : A$

S4 "necessarily" modality $\Box A$ = type of *closed* code of type A

(*e.g.* $\Box(\mathsf{nat} \to \mathsf{nat})$)

ensures safe evaluation ($\texttt{run} : \Box A \to A$)

but only closed code

A type system for staged computations?

$\lambda^{\Box}$ (Davies & Pfenning, 1996) $\quad \boxed{\Delta;\Gamma \vdash M : A}$

S4 "necessarily" modality $\Box A$ = type of *closed* code of type A

(*e.g.* $\Box(\mathsf{nat} \to \mathsf{nat})$)

ensures safe evaluation ($\mathtt{run} : \Box A \to A$)

but only closed code

$\lambda^{\bigcirc}$ (Davies, 1996) $\quad \boxed{\Gamma \vdash^n M : A}$

LTL "next" modality $\bigcirc A$ = type of *open* code

(*e.g.* $\bigcirc(\mathsf{nat} \to \mathsf{nat})$)

variables indexed by the current stage index n

ensures staged lexical scoping

but no safe code evaluation

A type system for staged computations?

$\lambda^{\square}$ (Davies & Pfenning, 1996) $\Delta;\Gamma \vdash M : A$

S4 "necessarily" modality $\square A$ = type of *closed* code of type A

(*e.g.* $\square(\mathsf{nat} \to \mathsf{nat})$)

ensures safe evaluation ($\texttt{run} : \square A \to A$)

but only closed code

$\lambda^{\bigcirc}$ (Davies, 1996) $\Gamma \vdash^{n} M : A$

LTL "next" modality $\bigcirc A$ = type of *open* code

(*e.g.* $\bigcirc(\mathsf{nat} \to \mathsf{nat})$)

variables indexed by the current stage index n

ensures staged lexical scoping

but no safe code evaluation

λ^{α} (Taha & Nielsen, 2003) $\Gamma \vdash^{\bar{\alpha}} M : A$

generalization of $\lambda^{\bigcirc}$ where stages are *named*

and can be *quantified over* (*e.g.* $\forall\alpha.\langle\mathsf{nat} \to \mathsf{nat}\rangle^{\alpha}$)

ensures safe evaluation & open code

but unclear logical foundation

Outline

In this talk

Close the gap between these systems:

- design a system λ^{ctx}
- show that it embeds them all

Contents

✓ Multi-staged programming by example
- Environment classifiers ($\lambda\bigcirc$, λ^{α})
- Contextual types ($\lambda\Box$, λ^{ctx})
- Translating λ^{α} to λ^{ctx}
- Summary and horizons

The λ-calculus $\lambda^{\rightarrow}$

(Church, 1940)

$$T, U ::= p \mid T \rightarrow U$$

$$E, F ::= x \mid \lambda x.E \mid E F$$

$$\Xi ::= \cdot \mid \Xi, x : T$$

$$\boxed{\Xi \vdash E : T}$$

VAR

$$\frac{(x : T) \in \Xi}{\Xi \vdash x : T}$$

LAM

$$\frac{\Xi, x : T \vdash E : U}{\Xi \vdash \lambda x.E : T \rightarrow U}$$

The temporal λ-calculus $\lambda\bigcirc$

(Davies, 1995)

$$T, U ::= p \mid T \to U \mid \bigcirc T$$
$$E, F ::= x \mid \lambda x. E \mid E\, F \mid \langle E \rangle \mid {\sim}E$$
$$\Xi ::= \cdot \mid \Xi, x :^n T$$

$$\boxed{\Xi \vdash^n E : T}$$

$$\textsc{Var}\ \frac{(x :^n T) \in \Xi}{\Xi \vdash^n x : T} \qquad \textsc{Lam}\ \frac{\Xi, x :^n T \vdash^n E : U}{\Xi \vdash^n \lambda x. E : T \to U} \qquad \textsc{Quote}\ \frac{\Xi \vdash^{n+1} E : T}{\Xi \vdash^n \langle E \rangle : \bigcirc T}$$

$$\textsc{Unquote}\ \frac{\Xi \vdash^n E : \bigcirc T}{\Xi \vdash^{n+1} {\sim}E : T}$$

The temporal λ-calculus $\lambda\bigcirc$

(Davies, 1995)

$$T, U ::= p \mid T \to U \mid \bigcirc T$$
$$E, F ::= x \mid \lambda x.E \mid E\,F \mid \langle E \rangle \mid {\sim}E$$
$$\Xi ::= \cdot \mid \Xi, x :^n T$$

$\boxed{\Xi \vdash^n E : T}$

$$\textsc{Var}\ \frac{(x :^n T) \in \Xi}{\Xi \vdash^n x : T} \qquad \textsc{Lam}\ \frac{\Xi, x :^n T \vdash^n E : U}{\Xi \vdash^n \lambda x.E : T \to U} \qquad \textsc{Quote}\ \frac{\Xi \vdash^{n+1} E : T}{\Xi \vdash^n \langle E \rangle : \bigcirc T}$$

$$\textsc{Unquote}\ \frac{\Xi \vdash^n E : \bigcirc T}{\Xi \vdash^{n+1} {\sim}E : T}$$

- $\texttt{run} : \bigcirc A \to A$ is unsafe
  ex: $\vdash \langle \lambda x. {\sim}(\texttt{run}\ \langle x \rangle) \rangle : \bigcirc(\bigcirc A \to A)$ but gets stuck

The environment classifiers λ-calculus λ^{α}

(Taha & Nielsen, 2003)

$$T, U ::= p \mid T \to U \mid \langle T \rangle^{\alpha}$$
$$E, F ::= x \mid \lambda x. E \mid E\, F \mid \langle E \rangle^{\alpha} \mid \sim E$$
$$\Xi ::= \cdot \mid \Xi, x :^{\bar{\alpha}} T$$

$$\boxed{\Xi \vdash^{\bar{\alpha}} E : T}$$

$$\textsc{Var}\ \frac{(x :^{\bar{\alpha}} T) \in \Xi}{\Xi \vdash^{\bar{\alpha}} x : T} \qquad \textsc{Lam}\ \frac{\Xi, x :^{\bar{\alpha}} T \vdash^{\bar{\alpha}} E : U}{\Xi \vdash^{\bar{\alpha}} \lambda x. E : T \to U} \qquad \textsc{Quote}\ \frac{\Xi \vdash^{\bar{\alpha}\alpha} E : T}{\Xi \vdash^{\bar{\alpha}} \langle E \rangle^{\alpha} : \langle T \rangle^{\alpha}}$$

$$\textsc{Unquote}\ \frac{\Xi \vdash^{\bar{\alpha}} E : \langle T \rangle^{\alpha}}{\Xi \vdash^{\bar{\alpha}\alpha} \sim E : T}$$

- $\texttt{run} : \bigcirc A \to A$ is unsafe
  ex: $\vdash \langle \lambda x. \sim(\texttt{run}\ \langle x \rangle) \rangle : \bigcirc(\bigcirc A \to A)$ but gets stuck

The environment classifiers λ-calculus λ^{α}

(Taha & Nielsen, 2003)

$$T, U ::= p \mid T \to U \mid \langle T \rangle^{\alpha} \mid \forall \alpha. T$$

$$E, F ::= x \mid \lambda x. E \mid E\,F \mid \langle E \rangle^{\alpha} \mid {\sim}E \mid \Lambda \alpha. E \mid E\,\alpha$$

$$\Xi ::= \cdot \mid \Xi, x :^{\bar{\alpha}} T$$

$$\boxed{\Xi \vdash^{\bar{\alpha}} E : T}$$

$$\textsc{Var}\;\frac{(x :^{\bar{\alpha}} T) \in \Xi}{\Xi \vdash^{\bar{\alpha}} x : T} \qquad \textsc{Lam}\;\frac{\Xi, x :^{\bar{\alpha}} T \vdash^{\bar{\alpha}} E : U}{\Xi \vdash^{\bar{\alpha}} \lambda x. E : T \to U} \qquad \textsc{Quote}\;\frac{\Xi \vdash^{\bar{\alpha}\alpha} E : T}{\Xi \vdash^{\bar{\alpha}} \langle E \rangle^{\alpha} : \langle T \rangle^{\alpha}}$$

$$\textsc{Unquote}\;\frac{\Xi \vdash^{\bar{\alpha}} E : \langle T \rangle^{\alpha}}{\Xi \vdash^{\bar{\alpha}\alpha} {\sim}E : T} \qquad \textsc{Gen}\;\frac{\Xi \vdash^{\bar{\alpha}} E : T \qquad \alpha \notin \mathrm{FV}(\Xi, \bar{\alpha})}{\Xi \vdash^{\bar{\alpha}} \Lambda \alpha. E : \forall \alpha. T} \qquad \textsc{Inst}\;\frac{\Xi \vdash^{\bar{\alpha}} E : \forall \beta. T}{\Xi \vdash^{\bar{\alpha}} E\,\alpha : T\{\alpha/\beta\}}$$

- `run` $: \bigcirc A \to A$ is unsafe
  ex: $\vdash \langle \lambda x. {\sim}(\texttt{run}\ \langle x \rangle) \rangle : \bigcirc(\bigcirc A \to A)$ but gets stuck

The environment classifiers λ-calculus λ^α

(Taha & Nielsen, 2003)

$$T, U ::= p \mid T \to U \mid \langle T \rangle^\alpha \mid \forall \alpha. T$$
$$E, F ::= x \mid \lambda x. E \mid E\, F \mid \langle E \rangle^\alpha \mid {\sim} E \mid \Lambda \alpha. E \mid E\, \alpha$$
$$\Xi ::= \cdot \mid \Xi, x :^{\bar{\alpha}} T$$

$\boxed{\Xi \vdash^{\bar{\alpha}} E : T}$

$$\textsc{Var}\ \frac{(x :^{\bar{\alpha}} T) \in \Xi}{\Xi \vdash^{\bar{\alpha}} x : T} \qquad \textsc{Lam}\ \frac{\Xi, x :^{\bar{\alpha}} T \vdash^{\bar{\alpha}} E : U}{\Xi \vdash^{\bar{\alpha}} \lambda x. E : T \to U} \qquad \textsc{Quote}\ \frac{\Xi \vdash^{\bar{\alpha}\alpha} E : T}{\Xi \vdash^{\bar{\alpha}} \langle E \rangle^\alpha : \langle T \rangle^\alpha}$$

$$\textsc{Unquote}\ \frac{\Xi \vdash^{\bar{\alpha}} E : \langle T \rangle^\alpha}{\Xi \vdash^{\bar{\alpha}\alpha} {\sim} E : T} \qquad \textsc{Gen}\ \frac{\Xi \vdash^{\bar{\alpha}} E : T \qquad \alpha \notin \mathrm{FV}(\Xi, \bar{\alpha})}{\Xi \vdash^{\bar{\alpha}} \Lambda \alpha. E : \forall \alpha. T} \qquad \textsc{Inst}\ \frac{\Xi \vdash^{\bar{\alpha}} E : \forall \beta. T}{\Xi \vdash^{\bar{\alpha}} E\, \alpha : T\{\alpha/\beta\}}$$

- $\texttt{run} : (\forall \alpha. \langle A \rangle^\alpha) \to \forall \alpha. A$ is safe
  ex: $\nvdash \langle \lambda x. {\sim}(\texttt{run}\ \langle x \rangle^\alpha) \rangle^\alpha$

The environment classifiers λ-calculus λ^{α}

Running code

$\texttt{run} : (\forall\alpha. \langle A\rangle^{\alpha}) \to \forall\alpha.A$

The environment classifiers λ-calculus λ^α

Running code

$\texttt{run} : (\forall\alpha.\langle A\rangle^\alpha) \to \forall\alpha.A$

Example (Two-level η-expansion)

$\lambda f.\langle\lambda x.\sim(f\,\langle x\rangle)\rangle : (\bigcirc p \to \bigcirc q) \to \bigcirc(p \to q)$

The environment classifiers λ-calculus λ^α

Running code

$\texttt{run} : (\forall\alpha.\langle A\rangle^\alpha) \to \forall\alpha.A$

Example (Two-level η-expansion)

$\lambda f.\langle\lambda x.\sim(f\,\langle x\rangle^\alpha)\rangle^\alpha : (\langle p\rangle^\alpha \to \langle q\rangle^\alpha) \to \langle p \to q\rangle^\alpha$

The environment classifiers λ-calculus λ^α

Running code

$\texttt{run} : (\forall \alpha . \langle A \rangle^\alpha) \to \forall \alpha . A$

Example (Two-level η-expansion)

$\lambda f . \Lambda \alpha . \langle \lambda x . \sim(f \, \alpha \, \langle x \rangle^\alpha) \rangle^\alpha : (\forall \alpha . \langle p \rangle^\alpha \to \langle q \rangle^\alpha) \to \forall \alpha . \langle p \to q \rangle^\alpha$

The environment classifiers λ-calculus λ^α

Running code

$\texttt{run} : (\forall\alpha.\langle A\rangle^\alpha) \to \forall\alpha.A$

Example (Two-level η-expansion)

$\lambda f.\Lambda\alpha.\langle\lambda x.\sim(f\,\alpha\,\langle x\rangle^\alpha)\rangle^\alpha : (\forall\alpha.\langle p\rangle^\alpha \to \langle q\rangle^\alpha) \to \forall\alpha.\langle p \to q\rangle^\alpha$

Issues

- what is the logical meaning of λ^α?
- what do α range over?
- complex operational semantics
  - syntax of value is context-sensitive (no BNF)
    $V^0 ::= \lambda x.V^0 \mid \langle V^1\rangle^\alpha$
  - 14 big-step rules

The λ-calculus $\lambda^{\to}$

(Church, 1940)

$$A, B ::= p \mid A \to B$$
$$M, N ::= x \mid \lambda x. M \mid M\, N$$

$$\boxed{\quad \Gamma \vdash M : A}$$

The modal λ-calculus λ□

(Davies & Pfenning, 1995)

$$A, B ::= p \mid A \to B \mid \Box A$$

$$M, N ::= x \mid \lambda x.\, M \mid M\, N \mid [M] \mid \text{let box } u = M \text{ in } N \mid u$$

$$\boxed{\Delta; \Gamma \vdash M : A}$$

$$\textsc{Box}\quad \frac{\Delta;\ \cdot \vdash M : A}{\Delta; \Gamma \vdash [M] : \Box A}$$

$$\textsc{LetBox}\quad \frac{\Delta; \Gamma \vdash M : \Box A \qquad \Delta, u :: \Box A\ ; \Gamma \vdash N : C}{\Delta; \Gamma \vdash \text{let box } u = M \text{ in } N : C}$$

$$\textsc{Meta}\quad \frac{u :: \Box A \in \Delta}{\Delta; \Gamma \vdash u : A}$$

The contextual λ-calculus λ_E^{ctx}

(Nanevski, Pfenning & Pientka, 2008)

$$A, B ::= p \mid A \to B \mid [\Psi . A]$$

$$M, N ::= x \mid \lambda x. M \mid M\, N \mid [\Psi . M] \mid \text{let box } u = M \text{ in } N \mid u\{\sigma\}$$

$\boxed{\Delta; \Gamma \vdash M : A}$

$$\textsc{Box}\ \frac{\Delta; \Psi \vdash M : A}{\Delta; \Gamma \vdash [\Psi . M] : [\Psi . A]}$$

$$\textsc{LetBox}\ \frac{\Delta; \Gamma \vdash M : [\Psi . A] \qquad \Delta, u :: [\Psi . A]; \Gamma \vdash N : C}{\Delta; \Gamma \vdash \text{let box } u = M \text{ in } N : C}$$

$$\textsc{Meta}\ \frac{u :: [\Psi . A] \in \Delta \qquad \Delta; \Gamma \vdash \sigma : \Psi}{\Delta; \Gamma \vdash u\{\sigma\} : A}$$

The contextual λ-calculus with first-class envs. λ_E^{ctx}

$$A, B ::= p \mid A \to B \mid [\Psi.A] \mid \forall \alpha.A$$

$$M, N ::= x \mid \lambda x.M \mid M\,N \mid [\Psi.M] \mid \text{let box } u = M \text{ in } N \mid u\{\sigma\} \mid \Lambda\alpha.M \mid M\,\Psi$$

$\boxed{\Delta;\Gamma \vdash M : A}$

$$\textsc{Box}\ \frac{\Delta;\Psi \vdash M : A}{\Delta;\Gamma \vdash [\Psi.M] : [\Psi.A]} \qquad \textsc{LetBox}\ \frac{\Delta;\Gamma \vdash M : [\Psi.A] \qquad \Delta, u :: [\Psi.A];\Gamma \vdash N : C}{\Delta;\Gamma \vdash \text{let box } u = M \text{ in } N : C}$$

$$\textsc{Meta}\ \frac{u :: [\Psi.A] \in \Delta \qquad \Delta;\Gamma \vdash \sigma : \Psi}{\Delta;\Gamma \vdash u\{\sigma\} : A}$$

$$\textsc{Gen}\ \frac{\Delta;\Gamma \vdash M : A \qquad \alpha \notin \mathrm{FV}(\Delta,\Gamma)}{\Delta;\Gamma \vdash \Lambda\alpha.M : \forall\alpha.A} \qquad \textsc{Inst}\ \frac{\Delta;\Gamma \vdash M : \forall\alpha.A}{\Delta;\Gamma \vdash M\,\Psi : A\{\alpha/\Psi\}}$$

The contextual λ-calculus with first-class envs. λ_E^{ctx}

$$
\begin{aligned}
A, B &::= p \mid A \to B \mid [\Psi.A] \mid \forall \alpha.A \\
M, N &::= x \mid \lambda x.M \mid M\,N \mid [\Psi.M] \mid \mathsf{let\,box}\; u = M \;\mathsf{in}\; N \mid u\{\sigma\} \mid \Lambda \alpha.M \mid M\,\Psi \\
\Gamma, \Psi &::= \alpha \mid \Gamma, x : A \\
\sigma &::= \mathsf{id}_\alpha \mid \sigma, x/M
\end{aligned}
$$

$\boxed{\Delta; \Gamma \vdash M : A}$

$$
\textsc{Box}\;\frac{\Delta; \Psi \vdash M : A}{\Delta; \Gamma \vdash [\Psi.M] : [\Psi.A]}
\qquad
\textsc{LetBox}\;\frac{\Delta; \Gamma \vdash M : [\Psi.A] \qquad \Delta, u :: [\Psi.A]; \Gamma \vdash N : C}{\Delta; \Gamma \vdash \mathsf{let\,box}\; u = M \;\mathsf{in}\; N : C}
$$

$$
\textsc{Meta}\;\frac{u :: [\Psi.A] \in \Delta \qquad \Delta; \Gamma \vdash \sigma : \Psi}{\Delta; \Gamma \vdash u\{\sigma\} : A}
$$

$$
\textsc{Gen}\;\frac{\Delta; \Gamma \vdash M : A \qquad \alpha \notin \mathrm{FV}(\Delta, \Gamma)}{\Delta; \Gamma \vdash \Lambda \alpha.M : \forall \alpha.A}
\qquad
\textsc{Inst}\;\frac{\Delta; \Gamma \vdash M : \forall \alpha.A}{\Delta; \Gamma \vdash M\,\Psi : A\{\alpha/\Psi\}}
$$

The contextual λ-calculus w/ first-class envs. λ_E^{ctx}

$$
\begin{aligned}
A, B &::= p \mid A \to B \mid [\Psi . A] \mid \forall \alpha . A \\
M, N &::= x \mid \lambda x . M \mid M\,N \mid [\Psi . M] \mid \mathsf{let\,box}\; u = M \;\mathsf{in}\; N \mid u\{\sigma\} \mid \Lambda \alpha . M \mid M\,\Psi \\
\Gamma, \Psi &::= \alpha \mid \Gamma, x : A \\
\sigma &::= \mathsf{id}_\alpha \mid \sigma, x/M
\end{aligned}
$$

Example

$$\lambda f . \Lambda \alpha . \mathsf{let\,box}\; u = f\,(\alpha, x : p)\,[\alpha, x . x] \;\mathsf{in}\; [\alpha . \lambda x . u\{\mathsf{id}_\alpha, x/x\}]$$

The contextual λ-calculus w/ first-class envs. λ_E^{ctx}

$$
\begin{aligned}
A, B &::= p \mid A \to B \mid [\Psi. A] \mid \forall \alpha. A \\
M, N &::= x \mid \lambda x. M \mid M\, N \mid [\Psi. M] \mid \mathsf{let\,box}\; u = M \;\mathsf{in}\; N \mid u\{\sigma\} \mid \Lambda \alpha. M \mid M\, \Psi \\
\Gamma, \Psi &::= \alpha \mid \Gamma, x : A \\
\sigma &::= \mathsf{id}_\alpha \mid \sigma, x/M
\end{aligned}
$$

Example

$$\lambda f. \Lambda \alpha. [\alpha. \lambda x. {\sim}(f\,(\alpha, x : p)\,[\alpha, x. x])\{\mathsf{id}_\alpha, x/x\}]$$

The contextual λ-calculus w/ first-class envs. λ_E^{ctx}

$$
\begin{aligned}
A, B &::= p \mid A \to B \mid [\Psi . A] \mid \forall \alpha . A \\
M, N &::= x \mid \lambda x. M \mid M\, N \mid [\Psi . M] \mid {\sim} M\{\sigma\} \qquad \mid \Lambda \alpha . M \mid M\, \Psi \\
\Gamma, \Psi &::= \alpha \mid \Gamma, x : A \\
\sigma &::= \mathsf{id}_\alpha \mid \sigma, x/M
\end{aligned}
$$

Example

$$\lambda f. \Lambda \alpha. [\alpha. \lambda x. {\sim}(f\,(\alpha, x : p)\,[\alpha, x. x])\{\mathsf{id}_\alpha, x/x\}]$$

The implicit contextual λ-calculus w/ first-class envs. λ_I^{ctx}

$$
\begin{aligned}
A, B &::= p \mid A \to B \mid [\Psi.A] \mid \forall \alpha.A \\
M, N &::= x \mid \lambda x.M \mid M\,N \mid [\Psi.M] \mid {\sim}M\{\sigma\} \qquad \mid \Lambda\alpha.M \mid M\,\Psi \\
\Gamma, \Psi &::= \alpha \mid \Gamma, x : A \\
\sigma &::= \mathsf{id}_\alpha \mid \sigma, x/M \\
\Sigma &::= \cdot \mid \Sigma; \Gamma
\end{aligned}
$$

$\boxed{\Sigma \vdash M : A}$

$$
\textsc{Box}\ \frac{\Sigma; \Psi \vdash M : A}{\Sigma \vdash [\Psi.M] : [\Psi.A]}
\qquad
\textsc{Unbox}\ \frac{\Sigma \vdash M : [\Psi.A] \qquad \Sigma; \Gamma \vdash \sigma : \Psi}{\Sigma; \Gamma \vdash {\sim}M\{\sigma\} : A}
$$

The implicit contextual λ-calculus w/ first-class envs. λ_I^{ctx}

Running code

- `run` $: (\forall \alpha. [\alpha. A]) \to \forall \alpha. A$

The implicit contextual λ-calculus w/ first-class envs. λ_I^{ctx}

Running code

- `run` $: (\forall \alpha.[\alpha.A]) \rightarrow \forall \alpha.A$
- `subst` $: \forall \alpha.[\alpha, x : A.B] \rightarrow [\alpha.A] \rightarrow [\alpha.B]$

The implicit contextual λ-calculus w/ first-class envs. λ_I^{ctx}

Running code

- $\texttt{run} : (\forall\alpha.[\alpha.A]) \to \forall\alpha.A$
- $\texttt{subst} : \forall\alpha.[\alpha, x : A.B] \to [\alpha.A] \to [\alpha.B]$

Example

$$\lambda f.\Lambda\alpha.[\alpha.\lambda x.\sim(f\,\alpha)\{\mathsf{id}_\alpha, x/x\}]$$
$$: (\forall\alpha.[\alpha, x : A.B]) \to \forall\alpha.[\alpha.A \to B]$$

Going contextual: from λ^{α} to λ_I^{ctx}

There is a transformation from λ^{α} to λ_I^{ctx}, specified as a judgment, and directed by the *derivation*:

$$\boxed{[\![\Xi \vdash^{\bar{\alpha}} E : T]\!] = \Sigma \vdash M : A}$$

Going contextual: from λ^α to λ_I^{ctx}

There is a transformation from λ^α to λ_I^{ctx}, specified as a judgment, and directed by the *derivation*:

$$\boxed{[\![\Xi \vdash^{\bar{\alpha}} E : T]\!] = \Sigma \vdash M : A}$$

Example

$$[\![\vdash^{\cdot} \lambda f.\, \Lambda\alpha.\, \langle \lambda x.\, {\sim}(f\, \alpha\, \langle x\rangle^\alpha)\rangle^\alpha : (\forall\alpha.\, \langle p\rangle^\alpha \to \langle q\rangle^\alpha) \to \forall\alpha.\, \langle p \to q\rangle^\alpha]\!] =$$
$$\vdash \lambda f.\, \Lambda\alpha.\, [\alpha.\, \lambda x.\, {\sim}(f\, \alpha\, [\alpha, x.\, x])\{\mathsf{id}_\alpha, x/x\}] :$$
$$(\forall\alpha.\, [\alpha.\, p] \to [\alpha.\, q]) \to \forall\alpha.\, [\alpha.\, p \to q]$$

Going contextual: from λ^{α} to λ_I^{ctx}

There is a transformation from λ^{α} to λ_I^{ctx}, specified as a judgment, and directed by the *derivation*:

$$\boxed{[\![\Xi \vdash^{\bar{\alpha}} E : T]\!] = \Sigma \vdash M : A}$$

Example

$$[\![\vdash^{\cdot} \Lambda\alpha.\, \lambda f.\, \langle \lambda x.\, \sim(f\, \langle x \rangle^{\alpha}) \rangle^{\alpha} : \forall\alpha.\, (\langle p \rangle^{\alpha} \to \langle q \rangle^{\alpha}) \to \langle p \to q \rangle^{\alpha}]\!] =$$
$$\vdash \Lambda\alpha.\, \lambda f.\, [\alpha.\, \lambda x.\, \sim(f\, [\alpha, x.\, x])\{\mathsf{id}_{\alpha}, x/x\}] :$$
$$\forall\alpha.\, ([\alpha, x : p.\, p] \to [\alpha, x : p.\, q]) \to [\alpha.\, p \to q]$$

Going contextual: from λ^α to λ_I^{ctx}

Definition (Type transformation)

$$\frac{}{[\![p]\!] = p} \qquad \frac{[\![T]\!] = T' \qquad [\![U]\!] = U'}{[\![T \to U]\!] = T' \to U'} \qquad \frac{[\![T]\!] = T'}{[\![\forall \alpha. T]\!] = \forall \alpha. T'}$$

$$\frac{[\![T]\!] = T'}{[\![\langle T \rangle^\alpha]\!] = [\Gamma(\alpha). T']}$$

Γ is *any* environment context; read it as a logic program, not a function.

Going contextual: from λ^α to λ_I^{ctx}

Definition (Derivation transformation)

$$\frac{\llbracket \Xi \vdash^{\bar{\alpha}\alpha} E : T \rrbracket = \Sigma; \Gamma(\alpha) \vdash M : A}{\llbracket \Xi \vdash^{\bar{\alpha}} \langle E \rangle^\alpha : \langle T \rangle^\alpha \rrbracket = \Sigma \vdash [\Gamma(\alpha).M] : [\Gamma(\alpha).A]}\ \textsc{Quote}{\to}\textsc{Box}$$

$$\frac{\llbracket \Xi \vdash^{\bar{\alpha}} E : \langle T \rangle^\alpha \rrbracket = \Sigma \vdash M : [\Psi(\alpha).A]}{\llbracket \Xi \vdash^{\bar{\alpha}\alpha} {\sim}E : T \rrbracket = \Sigma; \Psi(\alpha) \vdash {\sim}M\{\mathsf{id}_{\Psi(\alpha)}\} : A}\ \textsc{Unquote}{\to}\textsc{Unbox}$$

$$\frac{\llbracket \Xi \vdash^{\bar{\alpha}} E : \forall \alpha. T \rrbracket = \Sigma \vdash M : \forall \alpha. A}{\llbracket \Xi \vdash^{\bar{\alpha}} E\, \alpha : T \rrbracket = \Sigma \vdash M\, \Psi(\alpha) : A\{\alpha / \Psi(\alpha)\}}\ \textsc{Inst}{\to}\textsc{Inst}$$

$$\frac{}{\llbracket \Xi \vdash^{\bar{\alpha}} x : T \rrbracket = \llbracket \Xi \rrbracket_{\bar{\alpha}} \vdash x : \llbracket T \rrbracket}\ \textsc{Var}{\to}\textsc{Var}$$

Going contextual: from λ^α to λ_I^{ctx}

Theorem (Correctness)

If $\Xi \vdash^{\bar{\alpha}} E : T$ *and* $[\![\Xi \vdash^{\bar{\alpha}} E : T]\!] = \Sigma \vdash M : A$ *then:*

- $\Sigma \vdash M : A$ *holds*
- $[\![\Xi]\!]_{\bar{\alpha}} = \Sigma$
- $[\![T]\!] = A$.

Theorem (Decidability)

If $\Xi \vdash^{\bar{\alpha}} E : T$ *then* $\exists M$ *s.t.* $[\![\Xi \vdash^{\bar{\alpha}} E : T]\!] = [\![\Xi]\!]_{\bar{\alpha}} \vdash M : [\![T]\!]$.

Going explicit: from λ_I^{ctx} to λ_E^{ctx}

There is a higher-order transformation from λ_I^{ctx} to λ_E^{ctx}, close to *one-pass monadic normal form* transforms (Danvy, 2002):

$$\boxed{[\![\cdot]\!](\cdot) : M_I \to (M_I \to M_E) \to M_E}$$

Going explicit: from λ_I^{ctx} to λ_E^{ctx}

There is a higher-order transformation from λ_I^{ctx} to λ_E^{ctx}, close to *one-pass monadic normal form* transforms (Danvy, 2002):

$$\boxed{[\![\cdot]\!](\cdot) : M_I \to (M_I \to M_E) \to M_E}$$

Example

$$\begin{aligned}&[\![\lambda f.\,\Lambda\alpha.\,[\alpha.\,\lambda x.\,{\sim}(f\,(\alpha,x:p)\,[\alpha,x.x])\{\mathsf{id}_\alpha,x/x\}\,]]\!] = \\ &\qquad \lambda f.\,\Lambda\alpha.\,\mathsf{let\,box}\;u = f\,(\alpha,x:p)\,[\alpha,x.x]\;\mathsf{in}\;[\alpha.\,\lambda x.\,u\{\mathsf{id}_\alpha,x/x\}]\end{aligned}$$

Going explicit: from λ_I^{ctx} to λ_E^{ctx}

Definition (Term translation)

$$
\begin{aligned}
[\![x]\!](C) &= C(x) \\
[\![\lambda x. M]\!](C) &= [\![M]\!](\texttt{fn}\ M' \to C(\lambda x. M')) \\
[\![M\,N]\!](C) &= [\![M]\!](\texttt{fn}\ M' \to [\![N]\!](\texttt{fn}\ N' \to C(M'\,N'))) \\
[\![[\Gamma. M]]\!](C) &= C([\Gamma. [\![M]\!](\texttt{fn}\ M' \to M')]) \\
[\![\sim M\{\sigma\}]\!](C) &= [\![M]\!](\texttt{fn}\ M' \to [\![\sigma]\!](\texttt{fn}\ \sigma' \to \mathsf{let\,box}\ u = M'\ \mathsf{in}\ C(u\{\sigma'\})))
\end{aligned}
$$

Theorem (Correctness)

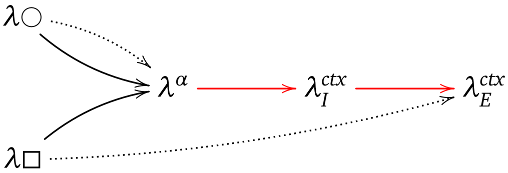
If $\Sigma \vdash M : A$ *then* $[\![\Sigma]\!] \vdash [\![M]\!](\texttt{fn}\ M \to M) : A$

To sum up…

λ^{α} variables annotated with stage

λ_I^{ctx} a stack of environments

λ_E^{ctx} two-zone presentation (validity & truth)

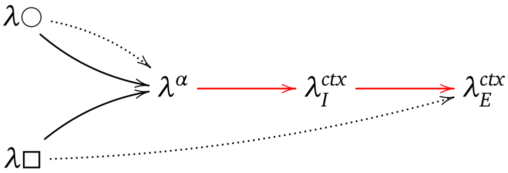


To sum up…

λ^{α} variables annotated with stage

λ_I^{ctx} a stack of environments

λ_E^{ctx} two-zone presentation (validity & truth)



The lessons

- $\lambda^{\alpha} \leq \lambda_E^{ctx}$
- λ_E^{ctx} has a simple operational semantics ($V ::= \lambda x.M \mid [\Psi.M]$)
- there *is* a two-zone presentation of LTL
  (just allow abstraction over environments)
- α range over "approximated" environments

A new look at Normalization

Normalization is evaluation of an annotated program:

$$\lambda x.\,((\lambda y.\,(y+1))\;x) \;:\; \mathsf{nat} \to \mathsf{nat}$$

A new look at Normalization

Normalization is evaluation of an annotated program:

$$\langle \lambda x. \sim((\lambda y. \langle \sim y + 1\rangle^{\alpha})\ \langle x\rangle^{\alpha})\rangle^{\alpha} \ :\ \langle \mathsf{nat} \to \mathsf{nat}\rangle^{\alpha}$$

A new look at Normalization

Normalization is evaluation of an annotated program:

$$[\alpha.\,\lambda x.\,{\sim}((\lambda y.\,[\alpha, x.\,{\sim}y\{\mathsf{id}_\alpha, x/x\} + 1])[\alpha, x.\,x])\{\mathsf{id}_\alpha, x/x\}] \\ :\ [\alpha.\,\mathsf{nat} \to \mathsf{nat}]$$

A new look at Normalization

Normalization is evaluation of an annotated program:

$$\mathsf{let\,box}\, u = (\lambda y.\, \mathsf{let\,box}\, v = y \;\mathsf{in}\; [\alpha, x.\, v\{\mathsf{id}_\alpha, x/x\} + 1])\; [\alpha, x.\, x]\; \mathsf{in}$$
$$[\alpha.\, \lambda x.\, u\{\mathsf{id}_\alpha, x/x\}] :\; [\alpha.\, \mathsf{nat} \to \mathsf{nat}]$$

A new look at Normalization

Normalization is evaluation of an annotated program:

$$\mathsf{let\,box}\, u = (\lambda y.\, \mathsf{let\,box}\, v = y \;\mathsf{in}\; [\alpha, x.\, v\{\mathsf{id}_\alpha, x/x\} + 1])\; [\alpha, x.\, x]\;\mathsf{in}$$
$$[\alpha.\, \lambda x.\, u\{\mathsf{id}_\alpha, x/x\}] : [\alpha.\, \mathsf{nat} \to \mathsf{nat}]$$

Conjecture (Staging/Binding-time Analysis)

If $M \longrightarrow^* V$ and V a normal form, then there is E s.t. $\texttt{run}\; E \Downarrow V$.

A new look at Normalization

Normalization is evaluation of an annotated program:

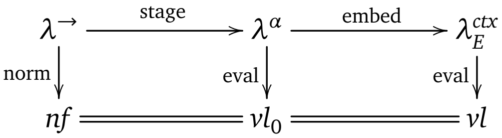
$$\mathsf{let\,box}\; u = (\lambda y.\, \mathsf{let\,box}\; v = y\; \mathsf{in}\; [\alpha, x.\, v\{\mathsf{id}_\alpha, x/x\} + 1])\; [\alpha, x.\, x]\; \mathsf{in}$$
$$[\alpha.\, \lambda x.\, u\{\mathsf{id}_\alpha, x/x\}] : \; [\alpha.\, \mathsf{nat} \rightarrow \mathsf{nat}]$$

Conjecture (Staging/Binding-time Analysis)

If $M \longrightarrow^* V$ and V a normal form, then there is E s.t. $\texttt{run}\; E \Downarrow V$.

Conjecture (Normalization by staged evaluation)

Staged evaluation "decomposes" normalization:



Conclusion

- contextual types as a logical foundation for staging?
  - strictly subsumes $\lambda\Box$, $\lambda\bigcirc$, $\lambda^\alpha\ldots$
  - in Curry-Howard correspondence with Contextual Logic
- technically: an embedding of environment classifiers into contextual types

Conclusion

- contextual types as a logical foundation for staging?
  - strictly subsumes $\lambda\Box$, $\lambda\bigcirc$, $\lambda^{\alpha}\ldots$
  - in Curry-Howard correspondence with Contextual Logic
- technically: an embedding of environment classifiers into contextual types

Future works

- relation to NbE?
- pattern-matching on code?
  ex: $\mathsf{case}\ (M : [\alpha.A])\ \mathsf{of}\ [\alpha.M] \to \ldots \mid [\alpha.\#p] \to \ldots$

Conclusion

- contextual types as a logical foundation for staging?
  - strictly subsumes $\lambda\Box$, $\lambda\bigcirc$, $\lambda^{\alpha}\ldots$
  - in Curry-Howard correspondence with Contextual Logic
- technically: an embedding of environment classifiers into contextual types

Future works

- relation to NbE?
- pattern-matching on code?
  ex: case $(M : [\alpha.A])$ of $[\alpha.M] \to \ldots \mid [\alpha.\#p] \to \ldots$

Thank you!