Contextual Types for Multi-Staged Programming

Matthias Puech[1]

Parsifal, Inria

Séminaire Gallium
Inria, Paris

February 29, 2016

[1] Joint work with Brigitte Pientka at McGill University

A bunch of inferences

« A bunch of inferences » is a noun phrase.

For any noun n,
« A bunch of $\sim$(pluraled n)» is a noun phrase.

« For any noun n,
« A bunch of $\sim$(pluraled n)» is a noun phrase.»
is valid a grammar rule.

For some part of speech P, « For any $\sim P \quad n$,
« A bunch of $\sim$(pluraled n)» is a $\sim P \quad$ phrase.»
is valid a grammar rule.

For some part of speech P, « For any $\sim P$ n,
« A bunch of $\sim$(pluraled n)» is a $\sim P$ phrase.»
is valid a grammar rule.

For some part of speech P, « For any $\sim P$ n,
« A bunch of $\sim$(pluraled n)» is a $\sim P$ phrase.»
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:

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

Motivation 1: Macros

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

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

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:

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

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:

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

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 ∗)*

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 *)*

⇝ 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 square (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 square (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 square (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 * square (square (x * square (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 square (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 * square (square (x * square (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.

$\rightsquigarrow$ 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:

$$e \;::=\; \ldots \;|\; «\, e \,» \;|\; \sim e \;|\; \texttt{run}\; e$$

Operational semantics:

$$\sim«\, e \,» \longrightarrow e \qquad\qquad \texttt{run}\; «\, e \,» \longrightarrow e$$

$$C \;::=\; \ldots \;|\; «\; \ldots \sim C \ldots \;»$$

Examples

Macros

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

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

```
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))»
```

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) »
```

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

$\textbf{let}\ \ \mathtt{e} = «\textbf{fun}\ x \rightarrow \sim((\textbf{fun}\ y \rightarrow «\ \sim y + 1\ ») \ «x»)»$

Examples

Full evaluation

$\textbf{let}\ \ \mathtt{e} = \text{«}\textbf{fun}\ \mathrm{x} \rightarrow \sim((\textbf{fun}\ \mathrm{y} \rightarrow \text{«}\ \sim\mathrm{y} + 1\ \text{»})\ \text{«}\mathrm{x}\text{»})\text{»}\ ;;$

$\textbf{val}\ \ \mathtt{e} = \text{«}\textbf{fun}\ \mathrm{x}\ \ \rightarrow \mathrm{x} + 1\text{»}$

Examples

Full evaluation

let `e` = «**fun** x → ∼((**fun** y → « ∼y + 1 ») «x»)» ;;
val `e` = «**fun** x → x + 1»

`run e 42`

Examples

Full evaluation

```
let e = «fun x → ~((fun y → « ~y + 1 ») «x»)» ;;
val e = «fun x → x + 1»

run e 42 ;;
- : int = 43
```

A type system for staged computations?

It must now ensure:

- lexical scoping (variables used in their binding context…)
- congruence (…at their binding stage) = "staged lexical scoping"
  ex: $\nvdash$ « **fun** x → ∼(x + 1) »
- evaluation of closed code
  ex: $\nvdash$ « **fun** x → ∼(run «x») »

Outline

Contents

- ✓ Multi-staged programming by example
- Contextual types (S4 $\rightarrow \lambda\Box\rightarrow \lambda^{ctx}\rightarrow \lambda^{ctx}_{I}$)
- Embedding environment classifiers (λ^{α})
- Consequences

Modal logic of necessity

$\Box A$ *A necessarily* true
(under no hypothesis)

Modal logic of necessity

$\Box A$ *A necessarily* true
(under no hypothesis)

Example

- $\Box A \supset A$
- $\Box A \supset \Box\Box A$
- $\Box(A \supset B) \supset (\Box A \supset \Box B)$

Modal logic of necessity

$\Box A$ $\quad$ *A necessarily* true
(under no hypothesis)

Example

- $\Box A \supset A$
- $\Box A \supset \Box\Box A$
- $\Box(A \supset B) \supset (\Box A \supset \Box B)$

Question

What is it in Curry-Howard correspondence with?

Modal logic of necessity

$\Box A$ — *A necessarily* true (under no hypothesis)

Example

- $\Box A \supset A$
- $\Box A \supset \Box\Box A$
- $\Box (A \supset B) \supset (\Box A \supset \Box B)$

Question

What is it in Curry-Howard correspondence with?

$\rightsquigarrow$ multi-staging (Davies & Pfenning, 1995)

$\Box A \ \approx\ $ « A »

The λ-calculus $\lambda^{\to}$

(Church, 1940)

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

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

The modal λ-calculus $\lambda\Box$

(Davies & Pfenning, 1995)

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

$$M, N ::= x \mid \lambda x.\, M \mid M\, N \mid [M] \mid \mathsf{let}_{\Box}\, u = M \,\mathsf{in}\, N \mid u$$

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

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

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

The contextual λ-calculus $\lambda^{[]}$

(Nanevski, Pfenning & Pientka, 2008)

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

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

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

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

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

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

(Puech, 2016?)

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

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

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

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

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

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

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

(Puech, 2016?)

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

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

$$\Gamma, \Psi ::= \alpha \mid \Gamma, x : A$$

$$\sigma ::= \mathsf{id}_\alpha \mid \sigma, x/M$$

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

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

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

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

Programming in λ^{ctx}

Expressive types

ex: $\forall \alpha.[\alpha.p \to q] \to [\alpha, p.\forall \beta.[\beta.q]]$

Programming in λ^{ctx}

Expressive types

ex: $\forall\alpha.[\alpha.p \to q] \to [\alpha,p.\forall\beta.[\beta.q]]$

Operational semantics

- Call by value + meta-variable substitution

Programming in λ^{ctx}

Expressive types

ex: $\forall\alpha.[\alpha.p \to q] \to [\alpha, p.\forall\beta.[\beta.q]]$

Operational semantics

- Call by value + meta-variable substitution
- $\texttt{run} : (\forall\alpha.[\alpha.A]) \to \forall\alpha.A$

Programming in λ^{ctx}

Expressive types

ex: $\forall\alpha.[\alpha.p \to q] \to [\alpha,p.\forall\beta.[\beta.q]]$

Operational semantics

- Call by value + meta-variable substitution
- $\texttt{run} : (\forall\alpha.[\alpha.A]) \to \forall\alpha.A$

$$\frac{M \downarrow \Lambda\alpha.[\alpha.N] \qquad N \downarrow V}{\texttt{run}\ M \downarrow \Lambda\alpha.V}$$

Programming in λ^{ctx}

Expressive types

ex: $\forall\alpha.[\alpha.p \to q] \to [\alpha,p.\forall\beta.[\beta.q]]$

Operational semantics

- Call by value + meta-variable substitution
- `run` $: (\forall\alpha.[\alpha.A]) \to \forall\alpha.A$

$$\frac{M \downarrow \Lambda\alpha.[\alpha.N] \qquad N \downarrow V}{\texttt{run}\ M \downarrow \Lambda\alpha.V}$$

- `subst` $: \forall\alpha.[\alpha,x:A.B] \to [\alpha.A] \to [\alpha.B]$

Programming in λ^{ctx}

Expressive types

ex: $\forall\alpha.[\alpha.p \to q] \to [\alpha, p.\forall\beta.[\beta.q]]$

Operational semantics

- Call by value + meta-variable substitution
- $\texttt{run} : (\forall\alpha.[\alpha.A]) \to \forall\alpha.A$

$$\frac{M \downarrow \Lambda\alpha.[\alpha.N] \qquad N \downarrow V}{\texttt{run}\ M \downarrow \Lambda\alpha.V}$$

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

$$\begin{aligned} &= \Lambda\alpha.\lambda xy.\,\mathsf{let}_\Box\, u = x \,\mathsf{in} \\ &\qquad \mathsf{let}_\Box\, v = y \,\mathsf{in}\, u\{\mathsf{id}_\alpha, v\{\mathsf{id}_\alpha\}\} \end{aligned}$$

Programming in λ^{ctx}

Expressive types

ex: $\forall\alpha.[\alpha.p \to q] \to [\alpha,p.\forall\beta.[\beta.q]]$

Operational semantics

- Call by value + meta-variable substitution
- $\texttt{run} : (\forall\alpha.[\alpha.A]) \to \forall\alpha.A$

$$\frac{M \downarrow \Lambda\alpha.[\alpha.N] \qquad N \downarrow V}{\texttt{run}\ M \downarrow \Lambda\alpha.V}$$

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

$$\begin{aligned} &= \Lambda\alpha.\lambda xy.\,\mathsf{let}_\Box\, u = x\,\mathsf{in} \\ &\qquad\qquad \mathsf{let}_\Box\, v = y\,\mathsf{in}\, u\{\mathsf{id}_\alpha, v\{\mathsf{id}_\alpha\}\}\end{aligned}$$

But heavy syntax

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

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

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

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

Going explicit: from λ_I^{ctx} back to λ^{ctx}

Example (1)

From:

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

Going explicit: from λ_I^{ctx} back to λ^{ctx}

Example (1)

From:

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

to:

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

Going explicit: from λ_I^{ctx} back to λ^{ctx}

Example (2)

From:

$$[\alpha.f\ [\beta.g\ \sim(h\ \sim x\{\mathsf{id}_\alpha\})\{\mathsf{id}_\beta\}]]$$

Going explicit: from λ_I^{ctx} back to λ^{ctx}

Example (2)

From:

$$[\alpha.f\ [\beta.g \sim\!(h \sim\! x\{\mathsf{id}_\alpha\})\{\mathsf{id}_\beta\}]]$$

to:

$$\mathsf{let}_\Box\, u = x\,\mathsf{in}\,[\alpha.f\ (\mathsf{let}_\Box\, v = h\ u\{\mathsf{id}_\alpha\}\,\mathsf{in}\,[\beta.g\ v\{\mathsf{id}_\beta\}])]$$

Going explicit: from λ_I^{ctx} back to λ^{ctx}

Example (2)

From:

$$[\alpha.f\ [\beta.g \sim(h \sim x\{\mathsf{id}_\alpha\})\{\mathsf{id}_\beta\}]]$$

to:

$$\mathsf{let}_\Box\, u = x\,\mathsf{in}\,[\alpha.f\ (\mathsf{let}_\Box\, v = h\ u\{\mathsf{id}_\alpha\}\,\mathsf{in}\,[\beta.g\ v\{\mathsf{id}_\beta\}])]$$

Theorem

If $\Sigma;\Gamma \vdash P : A$ then there exists Δ and M such that $\Delta;\Gamma \vdash M : A$.

Going explicit: from λ_I^{ctx} back to λ^{ctx}

Example (2)

From:

$$[\alpha.f\ [\beta.g\ \sim(h\ \sim x\{\mathsf{id}_\alpha\})\{\mathsf{id}_\beta\}]]$$

to:

$$\mathsf{let}_\Box\, u = x\,\mathsf{in}\,[\alpha.f\ (\mathsf{let}_\Box\, v = h\ u\{\mathsf{id}_\alpha\}\,\mathsf{in}\,[\beta.g\ v\{\mathsf{id}_\beta\}])]$$

Theorem

If $\Sigma;\Gamma \vdash P : A$ *then there exists* Δ *and* M *such that* $\Delta;\Gamma \vdash M : A$.

Proof.

Adapted from (Davies & Pfenning, 1995).
Multi-continuation one-pass monadic normal form. □

Outline

Contents

- ✓ Multi-staged programming by example
- ✓ Contextual types (S4 $\rightarrow \lambda\Box \rightarrow \lambda^{ctx} \rightarrow \lambda_I^{ctx}$)
- Embedding environment classifiers (λ^{α})
- Consequences

State of the art: Environment Classifiers λ^{α}

(Taha & Nielsen, 2003)

lineage of $\lambda\bigcirc$ (Davies, 1995)

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

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

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

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

The environment classifiers λ-calculus λ^α

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

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?
- 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 environment classifiers λ-calculus λ^α

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?
- 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

Question

Is it comparable to λ^{ctx}?

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

Example

From:

$$\lambda f.\,\Lambda\alpha.\,\langle\lambda x.\,{\sim}(f\,\alpha\,\langle x\rangle^\alpha)\rangle^\alpha$$

$$:(\forall\alpha.\,\langle p\rangle^\alpha \to \langle q\rangle^\alpha) \to \forall\alpha.\,\langle p \to q\rangle^\alpha$$

To:

Properties

- translates terms *and* types $\langle A\rangle^\alpha \rightsquigarrow [\Gamma.A]$

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

Example

From:

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

To:

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

Properties

- translates terms *and* types $\langle A\rangle^\alpha \rightsquigarrow [\Gamma.A]$
- environment information in target term/type
  $\rightsquigarrow$ translation on typing derivations

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

Example

From:

$$\lambda f.\, \Lambda\alpha.\, \langle \lambda x.\, {\sim}(f\, \alpha\, \langle x\rangle^\alpha)\rangle^\alpha$$

$$: (\forall\alpha.\, \langle p\rangle^\alpha \to \langle q\rangle^\alpha) \to \forall\alpha.\, \langle p \to q\rangle^\alpha$$

To:

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

$$: (\forall\alpha.\, [\alpha, p.\, p] \to [\alpha, p.\, q]) \to \forall\alpha.\, [\alpha.\, p \to q]$$

Properties

- translates terms *and* types $\langle A\rangle^\alpha \rightsquigarrow [\Gamma.\, A]$
- environment information in target term/type
  $\rightsquigarrow$ translation on typing derivations
- several possible translations
  $\rightsquigarrow$ outputs constrained *schemas*, containing *logic variables*

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

Example

From:

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

To:

$$\lambda f.\, \Lambda\alpha.\, [\alpha.\, \lambda x.\, \sim(f\ g_3(\alpha)\ [\alpha, x.x])\{\mathrm{id}_\alpha, x\}] :$$
$$(\forall\alpha.\, [g_1(\alpha).p] \to [g_2(\alpha).q]) \to \forall\alpha.\, [\alpha.p \to q]\ /$$
$$(g_1(g_3(\alpha)) = \alpha, x : p) \wedge (g_2(g_3(\alpha)) = \alpha, x : p)$$

Properties

- translates terms *and* types $\langle A\rangle^\alpha \rightsquigarrow [\Gamma.A]$
- environment information in target term/type
  $\rightsquigarrow$ translation on typing derivations
- several possible translations
  $\rightsquigarrow$ outputs constrained *schemas*, containing *logic variables*

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

Definition (Type translation)

$$
\begin{aligned}
\llbracket p \rrbracket &= p \\
\llbracket T \to U \rrbracket &= \llbracket T \rrbracket \to \llbracket U \rrbracket \\
\llbracket \forall \alpha.\, T \rrbracket &= \forall \alpha.\, \llbracket T \rrbracket \\
\llbracket \langle T \rangle^\alpha \rrbracket &= [g(\alpha).\, \llbracket T \rrbracket] \qquad g \text{ fresh}
\end{aligned}
$$

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

Definition (Type translation)

$$
\begin{aligned}
[\![p]\!] &= p \\
[\![T \to U]\!] &= [\![T]\!] \to [\![U]\!] \\
[\![\forall \alpha.\, T]\!] &= \forall \alpha.\, [\![T]\!] \\
[\![\langle T\rangle^\alpha]\!] &= [g(\alpha).\, [\![T]\!]] \qquad g \text{ fresh}
\end{aligned}
$$

Definition (Environment translation)

$$
\begin{aligned}
(\Xi, x :^X T)|_X^\alpha &= \Xi|_X^\alpha, x :' [\![T]\!] \\
(\Xi, x :^Y T)|_X^\alpha &= \Xi|_X^\alpha \quad \text{if } X \neq Y \\
(\cdot)|_X^\alpha &= \alpha
\end{aligned}
\qquad
\begin{aligned}
[\![\Xi]\!]_{\cdot} &= \Xi|_{\cdot}^{\alpha_0} \\
[\![\Xi]\!]_{X\alpha} &= [\![\Xi]\!]_X ; \Xi|_{X\alpha}^{\alpha}
\end{aligned}
$$

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

Definition (Derivation transformation)

$$\boxed{[\![\Xi \vdash^X E : T]\!] = \Sigma \vdash M : A \;/\; C}$$

$$\textsc{Var}\;\frac{[\![\Xi]\!]_X = \Sigma;\Gamma \qquad x : A \in \Gamma}{[\![\Xi \vdash^X x : T]\!] = \Sigma;\Gamma \vdash x : A \;/\; \top}$$

$$\textsc{App}\;\frac{[\![\Xi \vdash^X E : T \to U]\!] = \Sigma \vdash P : A \to B \;/\; C \qquad [\![\Xi \vdash^X F : T]\!] = \Sigma' \vdash Q : A' \;/\; C'}{[\![\Xi \vdash^X E\,F : U]\!] = \Sigma \vdash P\,Q : B \;/\; C \land C' \land A = A' \land \Sigma = \Sigma'}$$

$$\textsc{Box}\;\frac{[\![\Xi \vdash^{X\alpha} E : T]\!] = \Sigma;\Gamma \vdash P : A \;/\; C}{[\![\Xi \vdash^X \langle E\rangle^{\alpha} : \langle T\rangle^{\alpha}]\!] = \Sigma \vdash [\hat{\Gamma}.P] : [\tilde{\Gamma}.A] \;/\; C}$$

$$\textsc{Unbox}\;\frac{[\![\Xi \vdash^X E : \langle T\rangle^{\alpha}]\!] = \Sigma \vdash P : [\tilde{G}(\alpha).A] \;/\; C \qquad \Xi|^{\alpha}_{X\alpha} = \Gamma}{[\![\Xi \vdash^{X\alpha} {\sim}E : T]\!] = \Sigma;\Gamma \vdash {\sim}P\{\mathsf{id}(\hat{\Gamma})\} : A \;/\; C \land G(\alpha) = \Gamma}$$

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

Theorem (Type soundness)

If $[\![\Xi \vdash^X E : T]\!] = \Sigma \vdash P : A \;/\; C$ *and* $C\rho$ *holds for some instantiation* ρ*, then* $(\Sigma \vdash P : A)\rho$.

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

Theorem (Type soundness)

If $[\![\Xi \vdash^X E : T]\!] = \Sigma \vdash P : A \ / \ C$ *and* $C\rho$ *holds for some instantiation* ρ*, then* $(\Sigma \vdash P : A)\rho$.

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

Theorem (Type soundness)
If $[\![\Xi \vdash^X E : T]\!] = \Sigma \vdash P : A \;/\; C$ *and* $C\rho$ *holds for some instantiation* ρ*, then* $(\Sigma \vdash P : A)\rho$.

Theorem (Correctness)
If $\Xi \vdash^X E : T$ *and* $[\![\Xi \vdash^X E : T]\!] = \Sigma \vdash P : A \;/\; C$ *then there exists* ρ *such that* $[\![T]\!]\rho = A$ *and* $[\![\Xi]\!]_X \rho = \Sigma$.

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

Theorem (Type soundness)
If $[\![\Xi \vdash^X E : T]\!] = \Sigma \vdash P : A \;/\; C$ *and* $C\rho$ *holds for some instantiation* ρ*, then* $(\Sigma \vdash P : A)\rho$.

Theorem (Correctness)
If $\Xi \vdash^X E : T$ *and* $[\![\Xi \vdash^X E : T]\!] = \Sigma \vdash P : A \;/\; C$ *then there exists* ρ *such that* $[\![T]\!]\rho = A$ *and* $[\![\Xi]\!]_X\rho = \Sigma$.

Theorem (Decidability)
If $\Xi \vdash^X E : T$ *then there exists* Γ, P, A, C *and* ρ *such that* $[\![\Xi \vdash^X E : T]\!] = \Gamma \vdash P : A \;/\; C$ *and* $C\rho$ *holds.*

To sum up…

λ^α variables annotated with stage $>$ $\lambda\bigcirc$

λ_I^{ctx} a stack of environments

λ^{ctx} two-zone presentation (validity & truth) $>$ $\lambda\Box$

$$\begin{tikzcd}
\lambda\bigcirc \arrow[dr] \arrow[dr, dotted, bend left] & & & \\
& \lambda^\alpha \arrow[r] & \lambda_I^{ctx} \arrow[r] & \lambda^{ctx} \\
\lambda\Box \arrow[ur] \arrow[urrr, dotted, bend right] & & &
\end{tikzcd}$$

Outline

Contents

- ✓ Multi-staged programming by example
- ✓ Contextual types (S4 $\rightarrow \lambda^{\Box} \rightarrow \lambda^{ctx} \rightarrow \lambda_I^{ctx}$)
- ✓ Embedding environment classifiers (λ^{α})
- Consequences

Towards Contextual OCaml?

```
let f t = «y. fun x → ~t (x+y)»
```

Towards Contextual OCaml?

$\textbf{let}\ \texttt{f t} = \text{«}\texttt{y}.\ \textbf{fun}\ \texttt{x} \rightarrow {\sim}\texttt{t (x+y)}\text{»}\ ;;$

$\textbf{val}\ \texttt{f :}\ \text{«}\gamma.\ \texttt{int} \rightarrow \alpha\text{»} \rightarrow \text{«}\gamma,\ \texttt{int}.\ \texttt{int} \rightarrow \alpha\text{»} = \textbf{<fun>}$

Towards Contextual OCaml?

$\textbf{let}\ \mathtt{f}\ \mathtt{t} = «\mathtt{y}.\ \textbf{fun}\ \mathtt{x} \rightarrow \sim\!\mathtt{t}\ (\mathtt{x}+\mathtt{y})»\ ;;$

$\textbf{val}\ \mathtt{f}\ \mathtt{:}\ «\gamma.\ \mathtt{int} \rightarrow \alpha» \rightarrow «\gamma,\ \mathtt{int}.\ \mathtt{int} \rightarrow \alpha» = \textbf{<fun>}$

$\mathtt{run}\ (\mathtt{subst}\ (\mathtt{f}\ «\textbf{fun}\ \mathtt{x} \rightarrow \mathtt{x} * \mathtt{x}»)\ «2»)\ 3$

Towards Contextual OCaml?

```
let f t = «y. fun x → ~t (x+y)» ;;
val f : «γ. int → α» → «γ, int. int → α» = <fun>

run (subst (f «fun x → x * x») «2») 3 ;;
- : 25
```

Towards Contextual OCaml?

```
let f t = «y. fun x → ~t (x+y)» ;;
val f : «γ. int → α» → «γ, int. int → α» = <fun>

run (subst (f «fun x → x ∗ x») «2») 3 ;;
− : 25
```

Future work

- type inference for environment variables and code type

Pattern-matching on code

- useful for code generation, optimization (e.g., Camlp4):

```
let rec wrap e = match e with
| «x» → x
| «~e1 ~e2» → «apply ~(wrap e1) ~(wrap e2)»
| «fun x → ~e» → «fun x → ~(wrap e)»

let count_app e = «let apply x y =
                       incr counter;
                       x y in
                     ~(wrap e)»
```

Pattern-matching on code

- useful for code generation, optimization (e.g., Camlp4):

```
let rec wrap e = match e with
| «x» → x
| «~e1 ~e2» → «apply ~(wrap e1) ~(wrap e2)»
| «fun x → ~e» → «fun x → ~(wrap e)»

let count_app e = «let apply x y =
                     incr counter;
                     x y in
                   ~(wrap e)»
```

- typing well-understood in Beluga:
  - patterns variables carry environments
  - exhaustiveness decidable

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:

$$\begin{aligned}&\mathsf{let}_{\Box}\, u = (\lambda y.\, \mathsf{let}_{\Box}\, v = y\, \mathsf{in}\, [\alpha, x.\, v\{\mathsf{id}_\alpha, x/x\} + 1])\, [\alpha, x.\, x]\, \mathsf{in}\\ &\qquad\qquad [\alpha.\, \lambda x.\, u\{\mathsf{id}_\alpha, x/x\}] : [\alpha.\, \mathsf{nat} \to \mathsf{nat}]\end{aligned}$$

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:

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

Conjecture (Staging/Binding-time Analysis)

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

Conjecture (Normalization by staged evaluation)

Staged evaluation "decomposes" normalization:

$$\begin{tikzcd}
\lambda^{\rightarrow} \arrow[r, "\text{stage}"] \arrow[d, "\text{norm}"'] & \lambda^{\alpha} \arrow[r, "\text{embed}"] \arrow[d, "\text{eval}"'] & \lambda^{ctx} \arrow[d, "\text{eval}"'] \\
nf \arrow[r, equal] & vl_0 \arrow[r, equal] & vl
\end{tikzcd}$$