

# **CALL-BY-NAME, CALL-BY-VALUE, CALL-BY-NEED, AND THE LINEAR LAMBDA CALCULUS**

INITIATION À LA RECHERCHE

---

Quentin Schroeder

MPRI - Université Paris-Cité

## Some Context

- Every programming language has an evaluation strategy
- Call by Need is used in Haskell
- Call by Value is used in OCaml

## Goal

Study evaluation strategies via the linear lambda calculus

## Why?

- found linearity is relevant when studying Call by Need
- noticed it also applies for other strategies

1. Call by Name, Call by Value
2. Linear Lambda Calculus
3. Results 1
4. Call by Let, Call by Need
5. Affine Lambda Calculus
6. Results 2
7. Conclusion

# SIMPLY TYPED LAMBDA CALCULUS (SYNTAX)

---

**Types**  $A, B ::= \text{basic types} \mid A \rightarrow B$

**Terms**  $M, N ::= V \mid M N$

**Values**  $V, W ::= x \mid \lambda x.t$

---

$\text{Id} \frac{}{x : A \vdash x : A}$

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

$\rightarrow -\text{Elim} \frac{\Gamma \vdash M : A \rightarrow B \quad \Delta \vdash N : A}{\Gamma, \Delta \vdash M N : B}$

---

**Call by Name** Evaluate the body first

**Call by Value** Evaluate the arguments first

---

# CALL BY NAME (EXAMPLE)

## Call by Name

$$(\lambda x. x + x) (3 + 3)$$

$$\rightsquigarrow (3 + 3) + (3 + 3)$$

$$\rightsquigarrow 6 + (3 + 3)$$

$$\rightsquigarrow 6 + 6$$

$$\rightsquigarrow 12$$

# CALL BY NAME (EXAMPLE)

## Call by Name

$$\begin{aligned} & (\lambda x. x + x) (3 + 3) \\ & \rightsquigarrow (3 + 3) + (3 + 3) \\ & \rightsquigarrow 6 + (3 + 3) \\ & \rightsquigarrow 6 + 6 \\ & \rightsquigarrow 12 \end{aligned}$$

## Call by Value

$$\begin{aligned} & (\lambda x. x + x) (3 + 3) \\ & \rightsquigarrow (\lambda x. x + x) 6 \\ & \rightsquigarrow 6 + 6 \\ & \rightsquigarrow 12 \end{aligned}$$

# LINEAR LAMBDA CALCULUS (TERMS, TYPES)

**Types**  $A, B ::= \text{basic types} \mid !A \mid A \multimap B$

**Terms**  $M, N ::= x \mid !M \mid \text{let } !x = M \text{ in } N \mid \lambda x.M \mid M N$

$$\text{Id} \frac{}{x : A \vdash x : A}$$

$$\text{Dereliction} \frac{\Gamma, x : A \vdash M : B}{\Gamma, !x : !A \vdash M : B}$$

$$\text{Contraction} \frac{\Gamma, !y : !A, !z : !A \vdash M : B}{\Gamma, !x : !A \vdash M[y := x, z := x] : B}$$

$$\text{Weakening} \frac{\Gamma \vdash M : B}{\Gamma, !x : !A \vdash M : B}$$

$$!-\text{Intro} \frac{! \Gamma \vdash M : A}{! \Gamma \vdash !M : !A}$$

$$!-\text{Elim} \frac{! \Gamma \vdash M : !A \quad \Delta, !x : !A \vdash N : B}{\Gamma, \Delta \vdash \text{let } !x = M \text{ in } N : B}$$



# LINEAR LAMBDA CALCULUS (REDUCTION RULES)

---

$(\beta_{\multimap}) \quad (\lambda x.M) N \rightsquigarrow M[x := N]$

$(\beta_!) \quad \text{let } !x = !M \text{ in } !N \rightsquigarrow N[x := M]$

$(! \multimap) \quad (\text{let } !x = L \text{ in } M) N \rightsquigarrow \text{let } !x = L \text{ in } (M N)$

$(!!): \quad \text{let } !y = (\text{let } !x = L \text{ in } M) \text{ in } N \rightsquigarrow \text{let } !x = L \text{ in } (\text{let } !y = M \text{ in } N)$

---



## Translation

$(-)^{\circ} : \text{Call by Name LC} \rightarrow \text{Linear LC}.$

$Z^{\circ} \equiv Z$ , where  $Z$  is a basic type

$(A \rightarrow B)^{\circ} \equiv (!A^{\circ}) \multimap B^{\circ}$

$x^{\circ} \equiv x$

$(\lambda x.M)^{\circ} \equiv \lambda y.\text{let } !x = y \text{ in } M^{\circ}$

$(M N)^{\circ} \equiv M^{\circ} !N^{\circ}$

$(x_1 : A_1, \dots, x_n : A_n)^{\circ} \equiv !x_1 : A_1^{\circ}, \dots, !x_n : A_n^{\circ}$

# INTERPRETING CALL BY VALUE

## Translation

$(-)^*$  : Call by Value LC  $\rightarrow$  Linear LC

$Z^+ \equiv Z$  ,where  $Z$  is a basic type

$A^* \equiv !A^+$

$(A \rightarrow B)^+ \equiv A^* \multimap B^*$

$V^* \equiv !V^+$  ,where  $Z$  is a Value type

$x^+ \equiv x$

$(\lambda x.M)^+ \equiv \lambda y.\text{let } !x = y \text{ in } M^*$

$(M N)^* \equiv (\text{let } !z = M^* \text{ in } z) N^*$

$(x_1 : A_1, \dots, x_n : A_n)^* \equiv !x_1 : !A_1^+, \dots, !x_n : !A_n^+$

## Definitions

- Confluence: we have normal forms
- Subject reduction: typed terms reduce to typed terms

## Translations

- $(-)^*$  and  $(-)^{\circ}$  are sound and preserve types
- $(-)^*$  and  $(-)^{\circ}$  preserve reductions

## Results

- Linear LC satisfies confluence and subject reduction
- Translations let us transfer these results

# CALL BY LET LC (TYPES, TERMS)

---

<b>Types</b>	$A, B ::= \text{basic types} \mid A \rightarrow B$
<b>Terms</b>	$M, N ::= V \mid M N \mid \text{let } x = M \text{ in } N$
<b>Values</b>	$V, W ::= x \mid \lambda x. t$

---

$$\text{Id} \frac{}{x : A \vdash x : A} \quad \text{Let} \frac{\Gamma \vdash M : A \quad \Gamma, x : A \vdash N : B}{\Gamma, \Delta \vdash \text{let } x = M \text{ in } N : B}$$

$$\text{Contraction} \frac{\Gamma, y : A, z : A \rightarrow M : B}{\Gamma, x : A \vdash M[y := x, z := x] : B} \quad \text{Weakening} \frac{\Gamma \vdash M : B}{\Gamma, x : A \vdash M : B}$$

$$\rightarrow -\text{Intro} \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x. M : A \rightarrow B} \quad \rightarrow -\text{Elim} \frac{\Gamma \vdash M : A \rightarrow B \quad \Delta \vdash N : A}{\Gamma, \Delta \vdash M N : B}$$

- (I)**  $(\lambda x.M) N \rightsquigarrow \text{let } x = N \text{ in } M$
- (V)**  $\text{let } x = V \text{ in } N \rightsquigarrow N[x := V]$  , where  $V$  is a value
- (C)**  $(\text{let } x = L \text{ in } M) N \rightsquigarrow \text{let } x = L \text{ in } (M N)$
- (A)**  $\text{let } x = (\text{let } y = L \text{ in } M) \text{ in } N \rightsquigarrow \text{let } x = L \text{ in } (\text{let } y = M \text{ in } N)$

# CALL BY NEED (LAZY EVALUATION)

---

<b>Types</b>	$A, B ::= \text{basic types} \mid A \rightarrow B$
<b>Terms</b>	$M, N ::= V \mid M N \mid \text{let } x = M \text{ in } N$
<b>Values</b>	$V, W ::= x \mid \lambda x. t$

---



$$\text{Id} \frac{}{x : A \vdash x : A} \quad \text{Let} \frac{\Gamma \vdash M : A \quad \Gamma, x : A \vdash N : B}{\Gamma, \Delta \vdash \text{let } x = M \text{ in } N : B}$$

$$\text{Contraction} \frac{\Gamma, y : A, z : A \vdash M : B}{\Gamma, x : A \vdash M[y := x, z := x] : B} \quad \text{Weakening} \frac{\Gamma \vdash M : B}{\Gamma, x : A \vdash M : B}$$

$$\rightarrow -\text{Intro} \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x. M : A \rightarrow B} \quad \rightarrow -\text{Elim} \frac{\Gamma \vdash M : A \rightarrow B \quad \Delta \vdash N : A}{\Gamma, \Delta \vdash M N : B}$$

- (I)**  $(\lambda x.M) N \rightsquigarrow \text{let } x = N \text{ in } M$
- (V)**  $\text{let } x = V \text{ in } N \rightsquigarrow N[x := M]$  , where  $V$  is a value
- (C)**  $(\text{let } x = L \text{ in } M) N \rightsquigarrow \text{let } x = L \text{ in } (M N)$
- (A)**  $\text{let } x = (\text{let } y = L \text{ in } M) \text{ in } N \rightsquigarrow \text{let } x = L \text{ in } (\text{let } y = M \text{ in } N)$
- (G)**  $\text{let } x = M \text{ in } N \rightsquigarrow N$  if  $x$  not free in  $N$

# AFFINE LAMBDA CALCULUS (SYNTAX)

**Types**  $A, B, C ::= \text{basic types} \mid !A \mid A \multimap B$

**Terms**  $M, N ::= x \mid !M \mid \text{let } !x = M \text{ in } N \mid \lambda x.M \mid M N$

$$\text{Id} \frac{}{x : A \vdash x : A}$$

$$\text{Dereliction} \frac{\Gamma, x : A \vdash M : B}{\Gamma, !x : !A \vdash M : B}$$

$$\text{Contraction} \frac{\Gamma, !y : !A, !z : !A \vdash M : B}{\Gamma, !x : !A \vdash M[y := x, z := x] : B}$$

$$\text{Weakening}_{\text{Aff}} \frac{\Gamma \vdash M : B}{\Gamma, x : A \vdash M : B}$$

$$! \text{-Intro} \frac{! \Gamma \vdash M : A}{! \Gamma \vdash !M : !A}$$

$$! \text{-Elim} \frac{! \Gamma \vdash M : !A \quad \Delta, !x : !A \vdash N : B}{\Gamma, \Delta \vdash \text{let } !x = M \text{ in } N : B}$$



- $(\beta_{\rightarrow}) : (\lambda x.M) N \rightsquigarrow M[x := N]$
- $(\beta_!) : \text{let } !x = !M \text{ in } !N \rightsquigarrow N[x := M]$
- $(! \rightarrow) : \text{let } !x = L \text{ in } M N \rightsquigarrow \text{let } !x = L \text{ in } M N$
- $(!!) : \text{let } !y = \text{let } !x = L \text{ in } M \text{ in } N \rightsquigarrow \text{let } !x = L \text{ in } \text{let } !y = M \text{ in } N$
- $(!Weakening) : \text{let } !x = M \text{ in } N \rightsquigarrow N \quad \text{if } x \text{ not free in } N$

# INTERPRETING CALL BY LET

## Translation

$(-)^{*let}$ : Call By Let LC  $\rightarrow$  Linear LC

$$Z^{+let} \equiv Z \text{ , where } Z \text{ is a basic type}$$

$$A^{*let} \equiv !A^{+let}$$

$$(A \rightarrow B)^{+let} \equiv A^{*let} \multimap B^{*let}$$

$$V^{*let} \equiv !V^{+let} \text{ , where } Z \text{ is a Value type}$$

$$x^{+let} \equiv x$$

$$(\lambda x.M)^{+let} \equiv \lambda y.\text{let } !x = y \text{ in } M^{*let}$$

$$(M N)^{*let} \equiv (\text{let } !z = M^{*let} \text{ in } z) N^{*let}$$

$$(\text{let } x = M \text{ in } N)^{*let} \equiv \text{let } !x = M^{*let} \text{ in } N^{*let}$$

$$(x_1 : A_1, \dots, x_n : A_n)^{*let} \equiv !x_1 : !A_1^{+let}, \dots, !x_n : !A_n^{+let}$$

# INTERPRETING CALL BY NEED

## Translation

$(-)^{*need}$ : Call by Need LC  $\rightarrow$  **Affine LC**

$$Z^{+need} \equiv Z, \text{ where } Z \text{ is a basic type}$$

$$A^{*need} \equiv !A^{+need}$$

$$(A \rightarrow B)^{+need} \equiv A^{*need} \multimap B^{*need}$$

$$V^{*need} \equiv !V^{+need}, \text{ where } Z \text{ is a Value type}$$

$$x^{+need} \equiv x$$

$$(\lambda x.M)^{+need} \equiv \lambda y. \text{let } !x = y \text{ in } M^{*need}$$

$$(M N)^{*need} \equiv (\text{let } !z = M^{*need} \text{ in } z) N^{*need}$$

$$(\text{let } x = M \text{ in } N)^{*need} \equiv \text{let } !x = M^{*need} \text{ in } N^{*need}$$

$$(x_1 : A_1, \dots, x_n : A_n)^{*need} \equiv !x_1 : !A_1^{+need}, \dots, !x_n : !A_n^{+need}$$

## Note

Observationally equivalent = cannot distinguish via results

## Translations

- $(-)^{*let}$  and  $(-)^{*need}$  are sound and preserve types
- $(-)^{*let}$  and  $(-)^{*need}$  preserve reductions

## Results

- Affine LC satisfies confluence and subject reduction

## Note

Observationally equivalent = cannot distinguish via results

## Translations

- $(-)^{*let}$  and  $(-)^{*need}$  are sound and preserve types
- $(-)^{*let}$  and  $(-)^{*need}$  preserve reductions

## Results

- Affine LC satisfies confluence and subject reduction
- Translations let us transfer these results

## Note

Observationally equivalent = cannot distinguish via results

## Translations

- $(-)^{*let}$  and  $(-)^{*need}$  are sound and preserve types
- $(-)^{*let}$  and  $(-)^{*need}$  preserve reductions

## Results

- Affine LC satisfies confluence and subject reduction
- Translations let us transfer these results
- Call by Let LC conservatively extends the Call by Value LC

# RESULTS

## Note

Observationally equivalent = cannot distinguish via results

## Translations

- $(-)^{*let}$  and  $(-)^{*need}$  are sound and preserve types
- $(-)^{*let}$  and  $(-)^{*need}$  preserve reductions

## Results

- Affine LC satisfies confluence and subject reduction
- Translations let us transfer these results
- Call by Let LC conservatively extends the Call by Value LC
- Call by Let LC is observationally equivalent to Call by Value LC

# RESULTS

## Note

Observationally equivalent = cannot distinguish via results

## Translations

- $(-)^{*let}$  and  $(-)^{*need}$  are sound and preserve types
- $(-)^{*let}$  and  $(-)^{*need}$  preserve reductions

## Results

- Affine LC satisfies confluence and subject reduction
- Translations let us transfer these results
- Call by Let LC conservatively extends the Call by Value LC
- Call by Let LC is observationally equivalent to Call by Value LC
- Call by Need LC is observationally equivalent to Call by Name LC



## Summary

Linear LC is a good model for studying evaluation strategies

## Summary

Linear LC is a good model for studying evaluation strategies

## Future Work

- Sum Types (hard)
- Recursion (very hard)
- eta rules (very hard)
- equality (very hard)

[MOTW95]



John Maraist, Martin Odersky, David N. Turner, and Philip Wadler, *Call-by-name, Call-by-value, Call-by-need, and the Linear Lambda Calculus*, Electron. Notes Theor. Comput. Sci. **1** (1995), 370–392.