Call-by-name, Call-by-value, Call-by-need, and the Linear Lambda Calculus

Initiation à la recherche

Quentin Schroeder

MPRI - Université Paris-Cité

Idea

Motivation

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

Overview

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	$A, B ::=$	basic types $\mid A \rightarrow B$
Terms	$M, N ::=$	$V \mid M\ N$
Values	$V, W ::=$	$x \mid \lambda x.t$

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

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

Call by Name	Evaluate the body first
Call by Value	Evaluate the arguments first

Call by Name (Example)

Call by Name

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

Call by Name (Example)

Call by Name

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

Call by Value

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

Linear Lambda Calculus (Terms, Types)

Types : $A, B, C ::=$ basic types $\mid\ !A \mid A \multimap B$

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

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

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

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

Linear Lambda Calculus (Reduction Rules)

$(\beta_{\multimap})$	$(\lambda x.M)\ N \rightsquigarrow M[x := N]$
$(\beta_{!})$	$\text{let } !x = !M \text{ in } !N \rightsquigarrow N[x := M]$
$(! \multimap)$	$\text{let } !x = L \text{ in } M\ N \rightsquigarrow \text{let } !x = L \text{ in } M\ N$
$(!!)$:	$\text{let } !y = \text{let } !x = L \text{ in } M \text{ in } N \rightsquigarrow \text{let } !x = L \text{ in let } !y = M \text{ in } N$

$(-)^\circ$: Call by Name LC $\to$ Linear LC.

$$\begin{aligned}
Z^\circ &\equiv Z \quad \text{,where Z is a basic type} \\
(A \to B)^\circ &\equiv (!A^\circ) \multimap B^\circ \\
x^\circ &\equiv x \\
(\lambda x.M)^\circ &\equiv \lambda y.\mathrm{let}\ !x = y\ \mathrm{in}\ M \\
(M\ N)^\circ &\equiv M^\circ\ !N^\circ \\
(x_1 : A_1, \ldots, x_n : A_n)^\circ &\equiv !x_1 : A_1^\circ, \ldots, !x_n : A_n^\circ
\end{aligned}$$

Interpreting Call by Value

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

$$
\begin{aligned}
Z^+ &\equiv Z \quad \text{,where Z is a basic type} \\
A^* &\equiv !A^+ \\
(A \to B)^+ &\equiv A^* \multimap B^* \\
V^* &\equiv !V^+ \quad \text{,where Z is a Value type} \\
x^+ &\equiv x \\
(\lambda x.M)^+ &\equiv \lambda y.\text{let } !x = y \text{ in } M^* \\
(M\ N)^* &\equiv (\text{let } !z = M^* \text{ in } z)\ N^* \\
(x_1 : A_1, \ldots, x_n : A_n)^* &\equiv !x_1 :!A_1^+, \ldots, !x_n :!A_n^+
\end{aligned}
$$

Results

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)

Types : $A, B, C ::=$ basic types $\mid A \to B$

Terms : $L, M, N ::= V \mid M\ N \mid \text{let } x = M \text{ in } N$

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

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

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

$$\to -\mathit{Intro}\ \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B} \qquad \to -\mathit{Elim}\ \frac{\Gamma \vdash M : A \to B \qquad \Delta \vdash N : A}{\Gamma, \Delta \vdash M\ N : B}$$

Call by Let Reductions

(I) $(\lambda x.M)\ N \rightsquigarrow \text{let } x = N \text{ in } M$

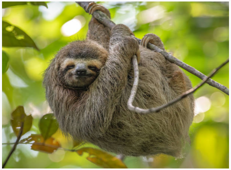
(V) $\text{let } x = V \text{ in } N \rightsquigarrow N[x := V]$, where V is a value

(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)

$$\textbf{Types}: A, B, C ::= \text{basic types} \mid A \to B$$
$$\textbf{Terms}: L, M, N ::= V \mid M\ N \mid \text{let } x = M \text{ in } N$$
$$\textbf{Values}: V, W ::= x \mid \lambda x.t$$



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

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

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

Call by Need Reductions

(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 : $A, B, C ::=$ basic types $\mid\ !A \mid A \multimap A$

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

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

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

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

Reduction For Affine LC

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

Interpreting Call by Let

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

$$
\begin{aligned}
Z^{+let} &\equiv Z \quad \text{,where Z is a basic type} \\
A^{*let} &\equiv !A^{+let} \\
(A \rightarrow B)^{+let} &\equiv A^{*let} \multimap B^{*let} \\
V^{*let} &\equiv !V^{+let} \quad \text{,where Z is a Value type} \\
x^{+let} &\equiv x \\
(\lambda x.M)^{+let} &\equiv \lambda y.\text{let } !x = y \text{ in } M^{*let} \\
(M\ N)^{*let} &\equiv (\text{let } !z = M^{*let} \text{ in } z)\ N^{*let} \\
(\text{let } x = M \text{ in } N)^{*let} &\equiv \text{let } !x = M^{*let} \text{ in } N^{*let} \\
(x_1 : A_1, \ldots, x_n : A_n)^{*let} &\equiv !x_1 : !A_1^{+let}, \ldots, !x_n : !A_n^{+let}
\end{aligned}
$$

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

$$
\begin{aligned}
Z^{+need} &\equiv Z \quad \text{,where Z is a basic type} \\
A^{*need} &\equiv !A^{+need} \\
(A \rightarrow B)^{+need} &\equiv A^{*need} \multimap B^{*need} \\
V^{*need} &\equiv !V^{+need} \quad \text{,where Z is a Value type} \\
x^{+need} &\equiv x \\
(\lambda x.M)^{+need} &\equiv \lambda y.\text{let } !x = y \text{ in } M^{*need} \\
(M\ N)^{*need} &\equiv (\text{let } !z = M^{*need} \text{ in } z)\ N^{*need} \\
(\text{let } x = M \text{ in } N)^{*need} &\equiv \text{let } !x = M^{*need} \text{ in } N^{*need} \\
(x_1 : A_1, \ldots, x_n : A_n)^{*need} &\equiv !x_1 : !A_1^{+need}, \ldots, !x_n : !A_n^{+need}
\end{aligned}
$$

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

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

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 linear lambda calculus

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 linear lambda calculus
- 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 linear lambda calculus
- Call by Let LC is observationally equivalent to Call by Value LC
- Call by Need LC is observationally equivalent to Call by Name LC

Conclusion

Summary

Linear LC is a good model for studying evaluation strategies

Conclusion

Summary

Linear LC is a good model for studying evaluation strategies

Future Work

- Product Types
- Sum Types (hard)
- Constants
- 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.