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

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

[MOTW95]

Overview

1. Simply Typed Lambda Calculus
2. Call by Name, Call by Value (Evaluation Strategies)
3. Linear Lambda Calculus
4. Interpretations of Call by Name, Call by Value
5. Call by Let, Call by Need
6. Affine Lambda Calculus
7. Interpretations of Call by Let, Call by Need
8. Results
9. Conclusion

Simply Typed Lambda Calculus (Syntax)

$$\textbf{Types}: A, B, C ::= \text{basic types} \mid A \to B$$

$$\textbf{Terms}: L, M, N ::= V \mid M\ N$$

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

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

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

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

Simply Typed Lambda Calculus (Evaluation Strategies)

Call by Name

Reduces on terms, not values

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

Strategy : Left-most, outermost redex first but no reduction under lambda.

(Redex = reducible expression, i.e. $(\lambda x.M)\ y$)

Simply Typed Lambda Calculus (Evaluation Strategies)

Call by Name

Reduces on terms, not values

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

Strategy : Left-most, outermost redex first but no reduction under lambda.

(Redex = reducible expression, i.e. $(\lambda x.M)\ y$)

Call by Value

Reduces on values, not terms

$(\beta_{value}) : (\lambda x.M)\ V \rightsquigarrow M[x := V]$

Strategy : Right-most, innermost redex first but no reduction under lambda.

Simply Typed Lambda Calculus (Evaluation Strategies)

Call by Name

Reduces on terms, not values

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

Strategy : Left-most, outermost redex first but no reduction under lambda.

(Redex = reducible expression, i.e. $(\lambda x.M)\ y$)

Call by Value

Reduces on values, not terms

$(\beta_{value}) : (\lambda x.M)\ V \rightsquigarrow M[x := V]$

Strategy : Right-most, innermost redex first but no reduction under lambda.

Note

We will shorten ”lambda calculus” as LC.

Example Call by Name

From [hb]

$$(\lambda p.\lambda q.p\ q\ p)\ ((\lambda a.\lambda b.a)\ (\lambda c.\lambda d.d))$$

$$\begin{aligned}
&(\lambda p.\lambda q.p\ q\ p)\ ((\lambda a.\lambda b.a)\ (\lambda c.\lambda d.d)) \\
&\quad \rightsquigarrow \lambda q.((\lambda a.\lambda b.a)\ (\lambda c.\lambda d.d))\ q\ ((\lambda a.\lambda b.a)\ (\lambda c.\lambda d.d)) \\
&\quad \rightsquigarrow \lambda q.(\lambda b.\lambda c.\lambda d.d)\ q\ ((\lambda a.\lambda b.a)\ (\lambda c.\lambda d.d)) \\
&\quad \rightsquigarrow \lambda q.(\lambda c.\lambda d.d)\ ((\lambda a.\lambda b.a)\ (\lambda c.\lambda d.d)) \\
&\quad \rightsquigarrow \lambda q.\lambda d.d
\end{aligned}$$

Example Call by Value

From [hb]

$$(\lambda p.\lambda q.p\ q\ p)\ ((\lambda a.\lambda b.a)\ (\lambda c.\lambda d.d))$$

$$\begin{aligned}
&(\lambda p.\lambda q.p\ q\ p)\ ((\lambda a.\lambda b.a)\ (\lambda c.\lambda d.d))\\
&\quad \rightsquigarrow (\lambda p.\lambda q.p\ q\ p)\ (\lambda b.\lambda c.\lambda d.d)\\
&\quad \rightsquigarrow \lambda q.(\lambda b.\lambda c.\lambda d.d)\ q\ (\lambda b.\lambda c.\lambda d.d)\\
&\quad \rightsquigarrow \lambda q.(\lambda c.\lambda d.d)\ (\lambda b.\lambda c.\lambda d.d)\\
&\quad \rightsquigarrow \lambda q.\lambda d.d
\end{aligned}$$

Linear Lambda Calculus (Terms, Types)

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

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

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

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

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

Linear Lambda Calculus (Reduction Rules)

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

Interpreting Call by Name

We define a translation mapping $(-)^\circ$ from the call by name LC to the linear LC.

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

We define a translation mapping $(-)^*$ from the call by value LC to the linear LC, together with a helper function $(-)^+$ on values

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

Call by let LC (Types, Terms)

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

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

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

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

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

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

Call by Let Reductions

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

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

(C) $(\text{let } x = L \text{ in } M)\ N \rightsquigarrow \text{let } x = L \text{ in } (M\ N)$

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

Call by Need

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

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 Need Reductions

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

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

(C) $(\text{let } x = L \text{ in } M)\ N \leadsto \text{let } x = L \text{ in } (M\ N)$

(A) $\text{let } x = (\text{let } y = L \text{ in } M) \text{ in } N \leadsto \text{let } x = L \text{ in } (\text{let } y = M \text{ in } N)$

(G) $\text{let } x = M \text{ in } N \leadsto N$ if x not free in N

Affine Lambda Calculus (Syntax)

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

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

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

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

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

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
- $(!Weakening)$: let $!x = M$ in $N \rightsquigarrow N$ if x not free in N

Interpreting Call by Let

We define a translation mapping $(-)^{*let}$ from the call by let LC to the linear LC, together with a helper function $(-)^{+let}$ on values

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

We define a translation mapping $(-)^{*need}$ from the call by need LC to the **affine LC**, together with a helper function $(-)^{+need}$ on values

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

Results

1. all the calculi introduced here are confluent and satisfy subject reduction

Results

1. all the calculi introduced here are confluent and satisfy subject reduction
2. all the translations are sound, preserve substitution, types and reductions

Results

1. all the calculi introduced here are confluent and satisfy subject reduction
2. all the translations are sound, preserve substitution, types and reductions
3. Call by Let LC conservatively extends the linear lambda calculus

Results

1. all the calculi introduced here are confluent and satisfy subject reduction
2. all the translations are sound, preserve substitution, types and reductions
3. Call by Let LC conservatively extends the linear lambda calculus
4. Call by Let LC is observationally equivalent to Call by Value LC

Results

1. all the calculi introduced here are confluent and satisfy subject reduction
2. all the translations are sound, preserve substitution, types and reductions
3. Call by Let LC conservatively extends the linear lambda calculus
4. Call by Let LC is observationally equivalent to Call by Value LC
5. Call by Need LC is observationally equivalent to Call by Name LC

Extensions

- Product Types - Yes
- Sum Types - Yes (with some caveats)
- Constants - Yes
- Recursion - Unclear for the Linear LC

Conclusion

- Linear LC is a good model for studying evaluation strategies
- Open questions remain: eta rules, equality

- Andrej Bauer (https://cs.stackexchange.com/users/1329/andrej bauer), *Lambda calculus - call-by-name and call-by-value reduction*, Computer Science Stack Exchange, URL:https://cs.stackexchange.com/q/101672 (version: 2018-12-17).
- 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.