CALL-BY-NAME EVALUATION OF RPC CALCULUS

Keishi Watanabe * and Shin-Ya Nishizaki **

* ** Department of Computer Science, Tokyo Institute of Technology, Tokyo, Japan
Email:* keishi.watanabe@lambda.cs.titech.ac.jp ** nisizaki@cs.titech.ac.jp

ABSTRACT: A remote procedure call, abbreviated RPC, is a communication mechanism between distributed computers connected through a network, which provides an interface similar to a procedure call in a single computer. Cooper and Wadler proposed the RPC calculus which formalizes the remote procedure call in the framework of the lambda calculus. The RPC calculus is an extension of the call-by-value lambda calculus by adding the notion of location. In this paper, we give a call-by-name evaluation strategy to the RPC calculus and investigate theoretical properties of the call-by-name RPC calculus.

KEYWORDS: programming language theory, functional programming, lambda calculus, operational semantics, remote procedure call, RPC calculus.

INTRODUCTION

A remote procedure call, abbreviated RPC, is a communication mechanism between distributed computers connected through a network, which provides an interface similar to a procedure call in a single computer. One of the famous implementation of RPC is Sun's RPC, which is used for Network File System on SunOS [1].

Cooper and Wadler [2] proposed the RPC calculus which formalizes the remote procedure call in the framework of the lambda calculus. The RPC calculus is an extension of the call-by-value lambda calculus by adding the notion of location. In this paper, we give a call-by-name evaluation strategy to the RPC calculus and investigate theoretical properties of the call-by-name RPC calculus.

An evaluation strategy [3], such as call-by-value and call-by-name, gives an order of evaluation among caller and callees in a function call. In lambda calculus, evaluation strategies can be defined as a big-step semantics. Call-by-value evaluation strategy is formalized as a binary relation $(-) \Downarrow (-)$ inductively-defined by the following rules.

$$\frac{}{V \Downarrow V} \qquad \frac{L \Downarrow \lambda x.N \quad M \Downarrow W \quad N[x{:=}W] \Downarrow V}{(L\ M) \Downarrow V}$$

On the other hand, the call-by-name evaluation strategy is formulated as follows.

$$\frac{}{V \Downarrow V} \qquad \frac{L \Downarrow \lambda x.N \quad N[x{:=}M] \Downarrow V}{(L\ M) \Downarrow V}$$

Symbols V and W represent values, which mean results of evaluation; a value is defined as either a variable or a lambda abstraction $\lambda x.M$. The first argument of the binary relation $\Downarrow$ means an expression to be evaluated and the second a result of the evaluation.

Since the binary relation $\Downarrow$ gives an evaluation result directly, we call it a big-step semantics.

Another style of formalization of the evaluation strategy is a small-step semantics, which is defined by reduction and reduction context dependent on each evaluation strategy. For example, the evaluation contexts of the call-by-value evaluation, are given by the following grammar.

$$E[\,] ::= [\,] \mid (E[\,]\ M) \mid (V\ E[\,])$$

The call-by-value reduction is defined inductively by the following rule.

$$E[(\lambda x.M)V] \rightarrow E[M[x := V]]$$

The call-by-name evaluation is given by the following evaluation context and the rule.

$$E[\,] ::= [\,] \mid (E[\,]\ M)$$

Cooper and Wadler [2] extends the Plotkin's style of the evaluation strategy for traditional function call to the one for remote procedure call; they proposed a lambda calculus for remote procedure calls, called the RPC calculus. The big-step semantics is given as a evaluation relation $M \Downarrow_l V$ inductively-defined by the following rules.

$$\frac{}{V \Downarrow_l V}$$

$$\frac{L \Downarrow_a (\lambda^a x.N) \quad M \Downarrow_a W \quad N[x{:=}W] \Downarrow_b V}{(LM) \Downarrow_a V}$$

The evaluation relation $M \Downarrow_l V$ means that the result of evaluation of M at location l is value V. In this paper, we study call-by-name evaluation for the RPC calculus.

RELATED WORKS

Call-by-value evaluation have been appeared in many programming languages since 1950's, such as FORTRAN and LISP. Call-by-name evaluation was originally proposed in ALGOL 60[4]. Call-by-value and call-by-name evaluation strategies were formalized as the small-step semantics by Plotkin [3]. The big-step semantics was invented by Kahn [5]. The RPC calculus was proposed by Cooper and Wadler in the paper [2], which is an extended lambda calculus by introducing the notion of location. Its operational semantics is provided as a big-step semantics based on the call-by-value evaluation. Narita and Nishizaki[7] developed a parallel abstract machine for the RPC calculus.

CALL-BY-NAME RPC CALCULUS

First, we introduce the syntax of the call-by-name RPC calculus, based on Cooper and Walder's call-by-value RPC calculus. We assume Var and Loc to be countable sets of variables and of locations, respectively. We use meta-variables *x,y,z,…* for variables and a,b,c,l,… for locations.

Definition 1 (Expression and Value). An expression of RPCcbn is defined inductively by the following grammar.

$$\begin{aligned} M ::= & \; x \\ & \mid (\lambda^a x.M) \\ & \mid (M \; N) \\ & \mid (M)^a \end{aligned}$$

A value RPCcbn is an expression satisfying the following grammar.

$$V ::= x \mid (\lambda^a x.M)$$

The first three constructs are the same as Cooper and Wadler's; $(\lambda^a x.M)$ means a function whose body M is assumed to be evaluated at location a. The last construct $(M)^a$ means an eval form at location a. We will sometimes incorporate constants such as numerals 0,1,2,… and function symbols +, ×, …, in order to enrich examples of the expression.

The big-step semantics in the following is given by a ternary relation $M \Downarrow_a V$ among a term M, a location l, and a value V, which intuitively means that a result of evaluation of a term M at a location a. The relation $M \Downarrow_a V$ is called an evaluation relation.

$$\frac{}{V \Downarrow_a V} \textbf{ Value}$$

$$\frac{L \Downarrow_a (\lambda^b x.N) \quad N[x := (M)^a] \Downarrow_b V}{(LM) \Downarrow_b V} \textbf{ Beta}$$

$$\frac{L \Downarrow_a (\lambda^b x.N) \quad N[x := (M)^c] \Downarrow_b V}{(L(M)^c) \Downarrow_b V} \textbf{ BetaEval}$$

$$\frac{M \Downarrow_b V}{(M)^b \Downarrow_a V} \textbf{ Eval}$$

An evaluation context is a term containing a hole which indicates to be reduced under an evaluation strategy, in the small-step semantics. Since the following evaluation context is supposed to be used for the call-by-name evaluation, the hole in the evaluation context locates at the head of function applications.

Definition 3 (*Evaluation Context*). An evaluation context of RPCcbn is defined by inductively by the following grammar.

$$\begin{aligned} E[\ ] ::= & [\ ] \\ & \mid ([\ ]\ N) \end{aligned}$$

Definition 4 (*Small-Step Semantics*) We define a *small-step semantics* as a relation between pairs of an expression and a location$(M)^a$.

$$\begin{array}{rcll} (E[(\lambda^b x.M)N])^a & \rightarrow & (E^a[N[x := (M)^a]])^b & \textbf{Beta} \\ (E[(\lambda^b x.M)(N)^c])^a & \rightarrow & (E^a[N[x := (M)^c]])^b & \textbf{BetaEval} \\ (E[(M)^b])^a & \rightarrow & (E^a[M])^b & \textbf{Eval} \end{array}$$

Lemma 1. If $E[M] \Downarrow_b V$, then there is a value W satisfying that $M \Downarrow_b W$ and $E[W] \Downarrow_b V$.

Proof. This lemma is proven by induction on the structure of $E[\ ]$.

Base case: $E[\ ] = [\ ]$. Suppose that $[M] \Downarrow_b V$; if you take V as W, then $W \Downarrow_b V$ is trivial.

Step case 1: $E[\ ] = (E'[\ ]\ N)$. Suppose that

$$(E'[M]\ N) \Downarrow_a V.$$

Then, by rule **Beta** of the big step semantics,

$$E'[M] \Downarrow_a (\lambda^b x.M') \quad (1)$$
$$M'[x := (N)^a] \Downarrow_b V \quad (2)$$

By the induction hypothesis, we know that there is a term W satisfying that

$$M \Downarrow_a W, \quad (3)$$
$$E'[W] \Downarrow_a (\lambda^b x.M'). \quad (4)$$

From (1) and (4),

$$(E'[W]\ N) \Downarrow_a V$$

by rule **Beta** of the big-step semantics, that is, $E[W] \Downarrow_b V$.

Step case 2: $E[\ ] = (E'[\ ](N)^c)$. Suppose that

$$(E'[M]\ (N)^c) \Downarrow_a V.$$

Then, by rule **BetaEval** of the big-step semantics, we know

$$E'[M] \Downarrow_a (\lambda^b x.M') \quad (5)$$
$$M'[x := (N)^c] \Downarrow_b V \quad (6)$$

By the induction hypothesis, we know that there is a term W satisfying that

$$M \Downarrow_a W, \quad (7)$$
$$E'[W] \Downarrow_a (\lambda^b x.M'). \quad (8)$$

From (5) and (8),

$$(E'[W]\ (N)^c) \Downarrow_a V$$

by rule **BetaEval** of the big-step semantics, that is,

$$E[W] \Downarrow_a V$$

End of Proof.

Lemma 2. If $M \Downarrow_a W$ and $E[W] \Downarrow_a V$, then $E[M] \Downarrow_a V$.

Proof. This lemma is proved by induction on the structure of $E[\ ]$.
Base case: $E[\ ] = [\ ]$. If it is supposed that

$$M \Downarrow_a W \text{ and } E[W] \Downarrow_a V$$

Then we know that $V = W$ and therefore it is trivial that

$$E[W] \Downarrow_a V.$$

Step case 1: $E[\ ] = (E'[\ ]\ N)$. Suppose that

$$M \Downarrow_a W \quad (9)$$
$$(E'[W]\ N) \Downarrow_a V. \quad (10)$$

From (10), there is a term M satisfying that

$$E'[W] \Downarrow_a (\lambda^b x.M) \quad (11)$$
$$M[x := (N)^a] \Downarrow_b V \quad (12)$$

By the induction hypothesis, (9) and (11), we have

$$E'[M] \Downarrow_a (\lambda^b x.M). \quad (13)$$

From (13) and (12), it is derived that by rule **Beta** that

$$(E'[M]\ N) \Downarrow_a V, \quad \text{that is,}$$

$$E[M] \Downarrow_a V.$$

Step case 2: $E[\ ] = (E'[\ ]\ (N)^c)$. Suppose that

$$M \Downarrow_a W \quad (14)$$
$$(E'[W]\ (N)^c) \Downarrow_a V. \quad (15)$$

From (15), there is a term M satisfying that

$$E'[W] \Downarrow_a (\lambda^b x.M) \quad (16)$$
$$M[x := (N)^c] \Downarrow_b V \quad (17)$$

By the induction hypothesis, (14) and (16), we have

$$E'[M] \Downarrow_a (\lambda^b x.M). \quad (18)$$

From (18) and (17), it is derived by rule **BetaEval** that

$$(E'[M]\ (N)^c) \Downarrow_a V,$$

that is,

$$E[M] \Downarrow_a V.$$

End of Proof.

Lemma 3. If $E^a[M] \Downarrow_a V$, then $E[M] \Downarrow_a V$.

Proof. This lemma is proved by induction on the structure of $E[\ ]$.
Base case: $E[\ ] = [\ ]$. If you suppose that

$$E^a[M] \Downarrow_a V$$

then it is trivial that

$$E[M] \Downarrow_a V$$

Since $E^a[\ ] = [\ ]$.

Step case 1: $E[\ ] = (E'[\ ]\ N)$. Suppose that

$$(E'^a[M]\ (N)^a) \Downarrow_a V.$$

By rule **BetaEval**, we have

$$E'^a[M] \Downarrow_a (\lambda^b x.L), \tag{19}$$
$$L[x := (N)^a] \Downarrow_b V. \tag{20}$$

By the induction hypothesis and (19),

$$E'[M] \Downarrow_a (\lambda^b x.L). \tag{21}$$

From (20) and (21), we have

$$(E'[M]\ N) \Downarrow_a V$$

by rule **Beta**.

Step case 2: $E[\ ] = (E'[\ ]\ (N)^a)$. Suppose that

$$(E'^a[M]\ (N)^a) \Downarrow_a V.$$

By rule **BetaEval**, we have

$$E'^a[M] \Downarrow_a (\lambda^b x.L), \tag{22}$$
$$L[x := (N)^c] \Downarrow_b V. \tag{23}$$

By the induction hypothesis and (22),

$$E'[M] \Downarrow_a (\lambda^b x.L). \tag{24}$$

From (23) and (24), we have

$$(E'[M]\ (N)^c) \Downarrow_a V$$

By rule **BetaEval**.
End of Proof.

Theorem 1. If $(M)^a \rightarrow^n V\ (n \geq 0)$, then $M \Downarrow_a V$.

Proof. This theorem is proven by mathematical induction on *n*.
Base case: $n = 1$. Since $(V)^a \to V$, $M = V$. Then $V \Downarrow_a V$, that is, $M \Downarrow_a V$.
Step case: $n > 1$. Suppose that
$(M)^a \to (M')^{a'} \to^{n-1} V$.

We make case analysis on $(M')^{a'} \to^{n-1} V$.
Case 1: $(E[(\lambda^b x.N)M])^a \to (E^a[N[x := (M)^a]])$. Since

$$(E^a[N[x := (M)^a]]) \to^{n-1} V,$$

from the induction hypothesis, we have

$$E^a[N[x := (M)^a]] \Downarrow_b V.$$

By Lemma 1, we know that there is a value *W* satisfying that

$$N[x := (M)^a] \Downarrow_b W \tag{25}$$
$$E^a[W] \Downarrow_a V \tag{26}$$

Then, from (40), rule **Value** and **Beta**, it is derived that

$$(\lambda^b x.N)M \Downarrow_a W. \tag{27}$$

From (27) and (26), we know

$$E^a[(\lambda^b x.N)\ M] \Downarrow_a V$$

by Lemma 2. By Lemma 3, we know

$$E[(\lambda^b x.N)\ M] \Downarrow_a V.$$

Case 2: $(E[(M)^b])^a \to (E^a[M])^b$. Since

$$(E^a[M])^b \to^{n-1} V$$

we have

$$E^a[M] \Downarrow_b V. \tag{28}$$

By Lemma 1, there is a term *W* satisfying that

$$M \Downarrow_b W, \tag{29}$$
$$E^a[W] \Downarrow_b V. \tag{30}$$

From (29), it is derived that

$$(M)^b \Downarrow_a W, \tag{31}$$

by rule **Eval**. By Lemma 2, (30) and (31),

$$E^a[(M)^b] \Downarrow_a V.$$

By Lemma 3,

$$E[(M)^b] \Downarrow_a V.$$

End of Proof.

The converse of Theorem 1 also holds. Before we prove it, we show a lemma which will be used in its proof.

Lemma 4. If

$$(L)^a \to (L')^{a'}$$

then

$$(L\ M)^a \to (L'\ (M)^a)^{a'} \text{ and } (L\ (M)^c)^a \to (L'\ (M)^c)^{a'}.$$

Proof. We make case analysis on $(L)^a \to (L')^{a'}$.

Case 1: $(E[(\lambda^b x.N)M])^a \to \left(E^a[N[x := (M)^a]]\right)^b$.
It holds that

$$(E[(\lambda^b x.N)M]\ L)^a \to (E^a[N[x := (M)^a]]\ (L)^a)^b.$$

and

$$(E[(\lambda^b x.N)(M)^d]\ (L)^c)^a \to (E^a[N[x := (M)^d]]\ (L)^c)^b.$$

Case 2: $(E[(\lambda^b x.N)(M)^d])^a \to \left(E^a[N[x := (M)^d]]\right)^b$.
It holds that

$$(E[(\lambda^b x.N)M]\ L)^a \to (E^a[N[x := (M)^a]]\ (L)^a)^b.$$

and

$$(E[(\lambda^b x.N)M]\ (L)^c)^a \to (E^a[N[x := (M)^a]]\ (L)^c)^b.$$

Case 3: $(E[(M)^b])^a \to (E^a[M])^b$.
It holds that

$$(E[(M)^b]\ L)^a \to (E^a[M]\ (L)^c)^b$$

and

$$(E[(M)^b]\ (L)^d)^a \to (E^a[M]\ (L)^d)^b.$$

End of Proof.

Theorem 2. If $M \downarrow_a V$, then $(M)^a \to^n (V)^{a'}$ for $n \geq 0$.

Proof. This theorem is proven by induction on the structure of $M \Downarrow_a V$.

Base case: $V \Downarrow_a V$.
It is trivial that

$$(V)^a = (M)^a \to^0 (V)^a.$$

Step case 1:

$$\frac{L \Downarrow_a (\lambda^b x.N) \quad N[x := (M)^a] \Downarrow_b V}{(L\ M) \Downarrow_a V}$$

By the induction hypothesis, we have

$$(L)^a \rightarrow^{n'} (\lambda^b x.N)^{a'}, \tag{32}$$
$$(N[x := (M)^a])^b \rightarrow^{n''} (V)^{a''}. \tag{33}$$

From (40) and Lemma 4, it is derived that

$$(L\ M)^a \rightarrow^{n'} ((\lambda^b x.N)\ (M)^a)^{a'} \tag{34}$$

By rule **BetaEval** of the small-step semantics,

$$((\lambda^b x.N)\ (M)^a)^{a'} \rightarrow (N[x := (M)^a])^b. \tag{35}$$

From (38), (39), and (41),

$$\begin{array}{ll} & (L\ M)\ a \\ \rightarrow^{n'} & ((\lambda^b x.N)(M)^a)^{a'} \\ \rightarrow & (N[x := (M)^a])^b \\ \rightarrow^{n''} & (V)^{a''}. \end{array}$$

Step Case 2:

$$\frac{L \Downarrow_a (\lambda^b x.N) \quad N[x := (M)^c] \Downarrow_b V}{(L\ (M)^c) \Downarrow_a V}$$

By the induction hypothesis, we have

$$(L)^a \rightarrow^{n'} (\lambda^b x.N)^{a'}, \tag{36}$$
$$(N[x := (M)^c])^b \rightarrow^{n''} (V)^{a''}. \tag{37}$$

From (40) and Lemma 4, it is derived that

$$(L\ (M)^c)^a \rightarrow^{n'} ((\lambda^b x.N)\ (M)^c)^{a'} \tag{38}$$

By rule **BetaEval** of the small-step semantics,

$$((\lambda^b x.N)\ (M)^c)^{a'} \rightarrow (N[x := (M)^c])^b. \tag{39}$$

From (38), (39), and (41),

$$\begin{array}{ll} & (L\ (M)^c)\ a \\ \rightarrow^{n'} & ((\lambda^b x.N)(M)^c)^{a'} \\ \rightarrow & (N[x := (M)^c])^b \\ \rightarrow^{n''} & (V)^{a''}. \end{array}$$

Step case 3:

$$\frac{M \Downarrow_b V}{(M)^b \Downarrow_a V}$$

By the induction hypothesis,

$$(M)^b \rightarrow^{n'} (V)^{a'}. \tag{40}$$

By rule **Eval** of the small-step semantics,

$$((M)^b)^a \rightarrow (M)^b. \tag{41}$$

From (40) and (41),

$$((M)^b)^a \rightarrow (M)^b \rightarrow^{n'} (V)^{a'}.$$

End of Proof.

CONCLUSIONS

We studied a call-by-name evaluation of the RPC calculus, which is formalized as both the big-step and the small-step semantics. The former gives a semantics function and the latter a step-wise computation. We then investigated the equivalence between two semantics.

The call-by-name evaluation is adopted in Algol 60 [4] but its efficiency is not enough for practical usage. However, lazy evaluation of functional programming languages such as Haskell is formalized as call-by-need evaluation strategy [6]. Our future research target is design and formalization of call-by-need evaluation in the RPC calculus. We studied a parallel abstract machine for the call-by-value RPC calculus in a previous work [7]. It is interesting to design and study a parallel abstract machine for the call-by-name RPC calculus.

ACKNOWLEDGMENT

This work was supported by Grants-in-Aid for Scientific Research (C) (24500009).

REFERENCES

A. D. Birrell and B. J. Nelson, "*Implementing remote procedure calls,*" ACM Transactions on Computer Systems, vol. 2, no. 1, pp. 39–59, 1984.

E. Cooper and P. Wadler, "The RPC calculus," in *Proceedings of the 11th ACM SIGPLAN conference on Principles and Practice of Declarative Programming, PPDP'09*, 2009, pp. 231–242.

G. Plotkin, "Call-by-name, call-by-value, and the λ-calculus," *Theoretical Computer Science*, vol. 1, pp. 125–159, 1975.

J. W. Backus, F. L. Bauer, J. Green, C. Katz, J. McCarthy, A. J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois, J. H. Wegstein, A. van Wijngaarden, and M. Woodger, "Revised report on the algorithm language ALGOL 60," *Commun. ACM*, vol. 6, no. 1, pp. 1–17, Jan. 1963.

G. Kahn, "Natural semantics," in *STACS 87*, ser. Lecture Notes in Computer Science, vol. 247. ACM, 1987, pp. 22–39.

Z. M. Ariola, J. Maraist, M. Odersky, M. Felleisen, and P. Wadler, "A call-by-need lambda calculus," in *Proceedings of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL '95. New York, NY, USA: ACM, 1995, pp. 233–246.

K. Narita and S. Nishizaki, "A Parallel Abstract Machine for the RPC Calculus," *Informatics Engineering and Information Science,* ser. Communications in Computer and Information Science, Vol. 253, 2011, pp. 320—332.