Big-step and Small-Step Semantics of the Call-by-Name RPC Calculus

Keishi Watanabe*, Shin-ya Nishizaki
Tokyo Institute of Technology, 2-12-1-W8-69, Ookayama, Meguroku, Tokyo 152-8552, Japan.

* Corresponding author. Tel.: +81-3-5734-2772; email: nisizaki@cs.titech.ac.jp
Manuscript submitted February 15, 2017; accepted May 21, 2017.
doi: 10.17706/jsw.12.7.570-580

Abstract: A remote procedure call (RPC) is a network communication technique between distributed computers. RPC is more approachable than the other network communication techniques since a programmer can use it in a similar manner to a procedure call in a sequential program on a single CPU computer. Cooper and Wadler proposed the RPC calculus and formalized the remote procedure call in the style of the lambda calculus. They used the call-by-value evaluation strategy for the RPC calculus. We may say that the RPC calculus is an extension of the traditional call-by-value lambda calculus by attaching a location. In the previous work, we developed a big-step semantics of the call-by-name RPC calculus and studied the translation of the call-by-name RPC calculus into the call-by name RMI calculus, in order to show the expressive power of the RMI calculus. In this paper, we newly propose a small-step semantics of the call-by-name RPC calculus. We prove the equivalence between the small-step and big-step semantics.

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

1. 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 most famous RPC implementations is Sun's RPC, which is used for Network File System on SunOS [5].

Cooper and Wadler [6] 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 the theoretical properties of the call-by-name RPC calculus. An evaluation strategy [8], such as call-by-value and call-by-name, gives the order of evaluation among caller and callees in a function call. In lambda calculus, an evaluation strategies can be defined as a big-step semantics. A 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 that mean the 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 is the 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.

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

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

Cooper and Wadler [6] extend Plotkin's style of evaluation strategy for a traditional function call to the one for a remote procedure call; they propose a lambda calculus for remote procedure calls, called the RPC calculus. The big-step semantics is given as an 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, formalizing its small-step and big-step semantics. We started to study the call-by-name RPC in our previous work [2]. In the paper [2], we proposed the big-step semantics of the RPC and RMI calculi and studied the relationship between the two calculi. In this paper, we newly propose the small-step semantics of the call-by-name RPC calculus. The small-step semantics is an extension of the reduction of the lambda calculus.

2. Related Works

The call-by-value evaluation has appeared in many programming languages since the 1950s, 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 small-step semantics by Plotkin[8]. Big-step semantics was invented by Kahn[7]. The RPC calculus was proposed by Cooper and Wadler [6], which is an extended lambda calculus introducing the notion of location. Its operational semantics is provided as a big-step semantics based on the call-by-value evaluation.

3. 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 metavariables $x, y, z, \cdots$ for variables and $a, b, c, l, \cdots$ for locations.

Definition **1 (Expression, Value)** An expression of RPC_{CBN} is defined inductively by the following grammar

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

A value of RPC_{CBN} 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 $0, 1, 2, \cdots$ and function symbols $+, \times, \ldots$, 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 the term M, the location l, and the value V, which intuitively means that a result of evaluation of the term M at the location a. The relation $M \Downarrow_a V$ is called an evaluation relation.

Definition 2 (Big-Step Semantics) We define a big-step semantics $M \Downarrow_a V$ for call-by-name evaluation as a ternary relation among the term M, the location a and the value V by the following rules, inductively.

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

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

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 is located at the head of function applications.

Definition 3 (Evaluation Context) An evaluation *context* of RPC_{CBN} is inductively defined by the following grammar.

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

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

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

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 $E[M] \Downarrow_b V$. If you take V as W, then $W \Downarrow_b W$ and $E[W] \Downarrow_b V$.

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

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

Then, by Rule **Beta** of the big-step semantics,

$$E'[M] \Downarrow_a (\lambda^b x, M') \qquad (1)$$

$$M'[x := (N)^a] \Downarrow_b V \tag{2}$$

By the induction hypothesis, we know that there is a term W satisfying that
From (1) and (4),

$$M \Downarrow_a W, \tag{3}$$

$$E'[W] \Downarrow_a (\lambda^b x.M'). \tag{4}$$

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

By rule **Beta** of the big-step semantics, that is, $E[W] \Downarrow_a V$.
Step Case 2: $E[\ ] = (E'[\ ]\quad (N)^c)$. Suppose that

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

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

$$E'[M] \Downarrow_a (\lambda^b x.M') \tag{5}$$

$$M'[x := (N)^c] \Downarrow_b V \tag{6}$$

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

$$M \Downarrow_a W, \tag{7}$$

$$E'[W] \Downarrow_a (\lambda^b x.M'). \tag{8}$$

From (5) and (8),
By rule **BetaEval** of the big-step semantics, that is,
End of Proof.

$$E[W] \Downarrow_a V$$

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

$$M \Downarrow_a W \tag{9}$$

$$(E'[W]\ N) \Downarrow_a V. \tag{10}$$

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

$$E'[W] \Downarrow_a (\lambda^b x.M) \tag{11}$$

$$M[x := (N)^a] \Downarrow_b V \tag{12}$$

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

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

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

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

That is,

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

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

$$M \Downarrow_a W \tag{14}$$

$$(E'[W]\ (N)^c) \Downarrow_a V. \tag{15}$$

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

$$E'[W] \Downarrow_a (\lambda^b x.M) \tag{16}$$

$$M[x := (N)^c] \Downarrow_b V \tag{17}$$

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

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

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

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

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

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

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 (29) and (21), we have

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

By Rule **Beta.**
Step Case 2: $E[\,] = (E'[\,]\ (N)^c)$. 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 $(V)^a \rightarrow V, M = V$. Then $V \Downarrow_a V$, that is $M \Downarrow_a V$.

Step Case: $n > 0$. Suppose that

$$(M)^a \rightarrow (M')^{a'} \rightarrow^{n-1} V.$$

We conduct a case analysis on $(M')^{a'} \rightarrow^{n-1} V$.

Case 1: $(E[(\lambda^b x.N)M])^a \rightarrow \left(E^a[N[x := (M)^a]]\right)$. Since

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

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), it is derived that

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

by Lemma 2. By Lemma 3, we have

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

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

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

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),

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

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 conduct a 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

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

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

It holds that

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

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

$$(L)^a \to^{n'} (\lambda^b x.N)^{a'}, \tag{32}$$

$$\left(N[x := (M)^a]\right)^b \to^{n''} (V)^{a''}. \tag{33}$$

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

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

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

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

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

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

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

$$(L)^a \to^{n'} (\lambda^b x.N)^{a'}, \tag{36}$$

$$\left(N[x := (M)^c]\right)^b \to^{n''} (V)^{a''}. \tag{37}$$

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

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

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

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

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

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

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,

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

From (40) and (41),

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

End of Proof.

4. Conclusion

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 the two semantics.

The call-by-name evaluation is adopted in Algol 60 [4] but its efficiency is not sufficient for practical usage. However, lazy evaluation of functional programming languages such as Haskell is formalized as the call-by-need evaluation strategy [3]. Our future research target is design and formalization of a call-by-need evaluation in the RPC calculus.

The $\lambda\sigma$-calculus[1] is an improved version of the lambda calculus, in which the variable binding is handled more precisely than the traditional lambda calculus, based on the idea of explicit substitution. We will improve our call-by-name evaluation, introducing the explicit substitution.

In Paper [9], we proposed a parallel abstract machine for the call-by-value RPC calculus. Design of a parallel abstract machine for the call-by-name RPC calculus is one of the most interesting research direction of this work.

Acknowledgment

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

References

[1] Abadi, M., Cardelli, L., Curien, P. L., & Lévy, J. J. (1991). Explicit substitutions. *Journal of Functional Programming*, *1(4)*, 375–416.

[2] Araki, S., & Nishizaki, S. (2014). Call-by-name evaluation of RPC and RMI calculi. *Theory and Practice of Computation*.

[3] Ariola, Z. M., Maraist, J., Odersky, M., Felleisen, M., & Wadler, P. (1995). A call-by-need lambda calculus. *Proceedings of the 22Nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*.

[4] Backus, J. W., Bauer, F. L., Green, J., Katz, C., McCarthy, J., Perlis, A. J., Rutishauser, H., Samelson, K., Vauquois, B., Wegstein, J. H., Van, W. A., & Woodger, M. (1963). Revised report on the algorithm language algol 60.

[5] Birrell, A. D., Nelson, B. J.: Implementing remote procedure calls. ACM Transactions on Computer Systems 2(1), 39–59 (1984)

[6] Cooper, E., & Wadler, P. (2009). The RPC calculus. *Proceedings of the 11th ACM SIGPLAN Conference on Principles and Practice of Declarative Programming*.

[7] Kahn, G. (1987). Natural semantics. *Lecture Notes in Computer Science*, *247*, 22–39.

[8] Plotkin, G. (1975). Call-by-name, call-by-value, and the λ-calculus. *Theoretical Computer Science 1*, 125–159.

[9] Narita, K., & Nishizaki, S. (2011). A parallel abstract machine for the RPC calculus. *Informatics Engineering and Information Science*.

Keishi Watanabe received his bachelor's and master's degrees from Tokyo Institute of Technology in 2013 and 2015, respectively.

Shin-ya Nishizaki is a professor of computer science at Tokyo Institute of Technology, Japan, where he leads a research group on formal theory on software systems. He received his bachelor's, master's and doctorate degrees from Kyoto University, in mathematical science. Before joining Tokyo Institute of Technology in 1998, Dr. Nishizaki held appointments in computer science as Associate Professor at Chiba University for two years and Assistant Professor at Okayama University for two years.