

On incomparability of some procedural semantics of untyped functional programs

Gor Ghazaryan

Russian-Armenian University
Yerevan, Armenia

e-mail: gghazaryan@hotmail.com

ABSTRACT

In the paper [1] it is shown that procedural semantics that use interpretation algorithm active and interpretation algorithms based on left, right, full substitutions and reductions are not complete and are pairwise incomparable for programs with more than one equation. In this paper we prove that those procedural semantics are incomparable for programs with only one equation.

Keywords

Functional programming, procedural semantics, interpretation algorithms.

1. INTRODUCTION

This paper is about procedural semantics that use interpretation algorithms introduced in [1]. The author of [1] proves that procedural semantics using interpretation algorithm active and interpretation algorithms based on left, right, full substitutions and reductions are pairwise incomparable. In this paper we present the notion of incomparability of procedural semantics for programs with one equation. The aim of the paper is to achieve the same results for programs with only one equation.

2. DEFINITIONS USED AND PREVIOUS RESULTS

The main definitions and notations used in this paper can be found in [1], [2]. Let V be a countable set of variables.

Definition 2.1 The set of terms Λ is the least set satisfying the following conditions:

1. If $x \in V$, then $x \in \Lambda$;
2. If $t_1, t_2 \in \Lambda$, then $(t_1 t_2) \in \Lambda$;
3. If $x \in V$, and $t \in \Lambda$, then $(\lambda x.t) \in \Lambda$.

Let us give short notations for terms: the term $(\dots(t_1 t_2) \dots t_k)$ where $t_i \in \Lambda$, $i = 1, \dots, k$, $k > 1$, is denoted as $t_1 t_2 \dots t_k$ and the term $(\lambda x_1(\lambda x_2(\dots \lambda x_m.t) \dots))$, where $t \in \Lambda$, $x_j \in V$, is denoted as $\lambda x_1 x_2 \dots x_m.t$, $j = 1, \dots, m$, $m > 0$.

The notions of a free and bound occurrence of a variable in a term and the notion of a free variable of a term are introduced in a conventional way. The set of all free variables of a term t is denoted as $fv(t)$. A term that does not contain free variables is called closed. To show mutually different variables of interest x_1, x_2, \dots, x_n , $n \geq 1$, of a term t , the notation $t[x_1, x_2, \dots, x_n]$ is used. The notation $t[t_1, t_2, \dots, t_n]$ (or $t[x_1 := t_1, x_2 := t_2, \dots, x_n := t_n]$) denotes the term obtained by the simultaneous substitution of the

terms t_1, t_2, \dots, t_n for all free occurrences of the variables x_1, x_2, \dots, x_n respectively, into the term t . The notation $t\{x_{i_1}, x_{i_2}, \dots, x_{i_k}\}$ is used to denote a term t with the indication of all $k \geq 0$ free occurrences of variables x_1, x_2, \dots, x_n (from the left to the right), $x_{i_j} \in \{x_1, x_2, \dots, x_n\}$, $j = 1, \dots, k$. The term obtained from a term t as a result of the simultaneous substitution of the terms t_1, t_2, \dots, t_k for the first, second, etc., k the free occurrences of the variables x_1, x_2, \dots, x_n is denoted as $t\{t_1, t_2, \dots, t_k\}$. A substitution is said to be admissible if all free variables of the term being substituted remain free after substitutions. We will consider only admissible substitution. Terms t_1 and t_2 are said to be congruent (which is denoted as $t_1 \equiv t_2$) if one term can be obtained from the other by renaming bound variables. Congruent terms are considered identical.

The notion of β -reduction is the following:

$$\beta = \left\{ ((\lambda x.t[x])t')', t[x := t'] \mid t, t' \in \Lambda, x \in V \right\}.$$

The relations $\rightarrow_\beta, \rightarrow_{\rightarrow\beta}, =_\beta$ are defined in a standard way. In what follows, we will omit symbol β . The term $(\lambda x.t[x])t'$ is referred to as a β -redex (simply redex). A term not containing redexes is referred to as a β -normal form (simply normal form).

We will denote the set of all normal forms by NF and the set of all closed normal forms by NF^0 . A term t is said to have a normal form if there exists a term $t' \in NF$ such that $t = t'$.

Also, the following notations are introduced:

$\langle t_1, \dots, t_m \rangle \equiv \lambda x.x t_1 \dots t_m$, where $x \in V$, $t_i \in \Lambda$, $x \notin fv(t_i)$, $i=1, \dots, m$, $m \geq 1$,
 $U_i^m \equiv \lambda x_1 \dots x_m.x_i$, where $x_i \in V$, $k \neq j \implies x_k \neq x_j$, $k, j = 1, \dots, m$, $1 \leq i \leq m$, $m \geq 1$,
 $P_i^m \equiv \lambda x.x U_i^m$, where $x \in V$, $1 \leq i \leq m$, $m \geq 1$,
 $Y \equiv \lambda h.(\lambda x.h(x x))(\lambda x.h(x x))$ is a fixed-point combinatory, where $x, h \in V$.

Definition 2.2 An untyped functional program is a system of equations of the form

$$\begin{aligned} f_1 &= t_1[f_1 \dots f_m] \\ &\dots \\ f_m &= t_m[f_1 \dots f_m] \end{aligned} \tag{1}$$

where $f_i \in V$, $i \neq j \implies f_i \neq f_j$, $t_i[f_1 \dots f_m] \in \Lambda$, $fv(t_i[f_1 \dots f_m]) \subseteq \{f_1, \dots, f_m\}$, $i, j = 1, \dots, m$, $m \geq 1$.

The first equation of the system is considered to be the principal equation of the program.

We will denote the set of all untyped functional programs with one equation by $Prog$.

Let us consider the solution of system (1)

$$(\tau_1, \dots, \tau_m), \quad (2)$$

where $\tau_i \equiv P_i^m (Y(\lambda x. < t_1 [P_1^m x, \dots, P_m^m x], \dots, t_m [P_1^m x, \dots, P_m^m x] >, i = 1, \dots, m.$

The term τ_1 is said to be the principal component of solution (2) or the fixed-point semantics of program (1).

The term τ_i is said to be recursive if $f_i \in fv(\tau_i)$, $i = 1, \dots, m.$

Definition 2.3 Let P be a program (1) and τ_1 be the fixed-point semantics of program P . The set $Fix(P)$ corresponding to the fixed point semantics of program P will be defined in the following way:

$$Fix(P) = \{(v_1, \dots, v_k, t_0) \mid \tau_1 v_1 \dots v_k \rightarrow t_0, v_1, \dots, v_k, t_0 \in NF^0, k \geq 0\}.$$

Let us introduce the notion of an interpretation algorithm A . Having received a program P of form (1) and term X on its input, the algorithm A either terminates with the result $X' \in NF$, where $fv(X') \cap \{f_1, \dots, f_m\} = \emptyset$, or works endlessly. Interpretation algorithms use two type of operations:

(a) the substitution of terms $t_1 [f_1 \dots f_m], \dots, t_m [f_1 \dots f_m]$ for some free occurrences of variables f_1, \dots, f_m respectively, and

(b) a one step β -reduction.

We will denote the set of all interpretation algorithms by A .

Definition 2.4 Let P be a program (1) and A be an interpretation algorithm. The set $Proc_A(P)$ corresponding to the procedural semantics that uses the interpretation algorithm A will be defined in the following way:

$$Proc_A(P) = \{(v_1, \dots, v_k, t_0) \mid A(P, t_1 [f_1, \dots, f_m] v_1 \dots v_k) \text{ is determined and equal to } t_0, \text{ where } v_1, \dots, v_k, t_0 \in NF^0, k \geq 0\}.$$

3. INCOMPARABILITY OF PROCEDURAL SEMANTICS

We say that procedural semantics using interpretation algorithm A is consistent (complete) if $Proc_A(P) \subset Fix(P)$ ($Fix(P) \subset Proc_A(P)$, respectively) for any program P .

Procedural semantics that use interpretation algorithms A and B are incomparable for programs with one equation if

- (a) $\exists P \in Prog; Proc_A(P) \not\subset Proc_B(P)$,
- (b) $\exists P \in Prog; Proc_B(P) \not\subset Proc_A(P)$.

The following theorem is proved in [1]:

Theorem 3.1 (on consistency) For any program P and interpretation algorithm A , $Proc_A(P) \subset Fix(P)$.

Let $L(X)$ be the term X if $X \in NF$, and X' obtained from X by applying a one-step left reduction. We define algorithm N , when given term X , constructs its normal form if it exits or functions endlessly otherwise.

Algorithm N.

Input: term X .

Output: term $N(X)$ if N is determined on X .

- 1. If $X \in NF$, then X ; else $N(L(X))$.

Let us describe the set of interpretation algorithms $SNFR$ in which reduction to the normal form alternates with substitution operation. Each algorithm $S \in SNFR$ has the

following form:

Input: program P of form (1) and term X , where $fv(X) \subset \{f_1, \dots, f_m\}$,

Output: term $S(P, X)$ if S is determined on P and X .

- 1. $N(X)$ is not determined, then an infinite process corresponding to the endless functioning of N on X takes place; else go to item 2.

- 2. $N(X) \equiv t[f_1 \dots f_m] \in NF \setminus NF^0$, then $S(P, t')$, where t' is obtained from t by the simultaneous substitution of the terms $t_1 [f_1 \dots f_m], \dots, t_m [f_1 \dots f_m]$ for some free occurrences of the variables f_1, \dots, f_m respectively.

Let us describe the examples of $SNFR$ which are used in this paper.

Algorithm LSNFR (left substitution and reduction to the normal form).

Input: program P of form (1) and term X , where $fv(X) \subset \{f_1, \dots, f_m\}$,

Output: term $LSNFR(P, X)$ if $LSNFR$ is determined on P and X .

- 1. If $N(X)$ is not determined, then an infinite process corresponding to the endless functioning of N on X takes place; else go to item 2.

- 2. If $N(X) \equiv t\{f_{i_1}, \dots, f_{i_s}\} \in NF \setminus NF^0$, where $f_{i_j} \in \{f_1, \dots, f_m\}$, $j = 1, \dots, s$, $s \geq 1$, then $LSNFR(P, t\{t_{i_1}[f_1, \dots, f_m], f_{i_2}, \dots, f_{i_s}\})$, else $N(X)$.

Algorithm RSNFR (right substitution and reduction to the normal form).

Input: program P of form (1) and term X , where $fv(X) \subset \{f_1, \dots, f_m\}$,

Output: term $RSNFR(P, X)$ if $RSNFR$ is determined on P and X .

- 1. If $N(X)$ is not determined, then an infinite process corresponding to the endless functioning of N on X takes place; else go to item 2.

- 2. If $N(X) \equiv t\{f_{i_1}, \dots, f_{i_s}\} \in NF \setminus NF^0$, where $f_{i_j} \in \{f_1, \dots, f_m\}$, $j = 1, \dots, s$, $s \geq 1$, then $RSNFR(P, t\{f_{i_1}, f_{i_2}, \dots, t_{i_s}[f_1, \dots, f_m]\})$, else $N(X)$.

Algorithm FSNFR (full substitution and reduction to the normal form).

Input: program P of form (1) and term X , where $fv(X) \subset \{f_1, \dots, f_m\}$,

Output: term $FSNFR(P, X)$ if $FSNFR$ is determined on P and X .

- 1. If $N(X)$ is not determined, then an infinite process corresponding to the endless functioning of N on X takes place; else go to item 2.

- 2. If $N(X) \equiv t\{f_{i_1}, \dots, f_{i_s}\} \in NF \setminus NF^0$, where $f_{i_j} \in \{f_1, \dots, f_m\}$, $j = 1, \dots, s$, $s \geq 1$, then $FSNFR(P, t\{t_{i_1}[f_1, \dots, f_m], t_{i_2}[f_1, \dots, f_m], \dots, t_{i_m}[f_1, \dots, f_m]\})$, else $N(X)$.

Algorithm ACT

Input: program P of form (1) and term X , where $fv(X) \subset \{f_1, \dots, f_m\}$,

Output: term $ACT(P, X)$ if ACT is determined on P and X .

- 1. If $X \in NF$ and $fv(X) \cap \{f_1, \dots, f_m\} = \emptyset$, then X ; else go to item 2,

- 2. If $X \equiv X\{f_{i_1}, \dots, f_{i_s}\}$, where $f_{i_j} \in \{f_1, \dots, f_m\}$, $j = 1, \dots, s$, $s \geq 1$ and the occurrence f_{i_1} is on the left of the leftmost redex of the term X , then

$ACT(P, X\{t_{i_1}[f_1, \dots, f_m], f_{i_2}, \dots, f_{i_s}\})$; else go to item 3,

- 3. $X \equiv X(\lambda x.t)\tau$, where $x \in V$, $t, \tau \in \Lambda$ and $(\lambda x.t)\tau$ is the leftmost redex of term X , then if $x \in fv(t)$, then

$ACT(P, X_{t[x:=ACT(P, \tau)]})$, else $ACT(P, X_t)$.

Remark 3.1 We have changed the item 3 of the algorithm ACT defined in [1] to avoid working endless on recursive terms.

In [1] the author shows that the procedural semantics that use the interpretation algorithms *LSNFR*, *RSNFR*, *FSNFR*, *ACT* are pairwise incomparable for programs with more than one equation. In this paper we prove the following theorem:

Theorem 3.2 Interpretation algorithms *LSNFR*, *RSNFR*, *FSNFR*, *ACT* are pairwise incomparable for programs with one equation.

REFERENCES

- [1] S. Nigyan, S. Avetisyan, "On Procedural Semantics of Untyped Functional Programs", *Computer Science and Information Technologies*, 60-62, 2007.
- [2] H.P. Barendregt, "The lambda calculus, its syntax and semantics", *Amsterdam: North-Holland Pub. Comp.*, 1981.