PROCEEDINGS OF THE YEREVAN STATE UNIVERSITY

*Physical and Mathematical Sciences* 2018, **52**(2), p. 119–133

*Informatics*

# ON INTERPRETATION OF TYPED AND UNTYPED FUNCTIONAL PROGRAMS

S. A. NIGIYAN [*]

*Chair of Programming and Information Technologies YSU, Armenia*

In this paper the interpretation algorithms for typed and untyped functional programs are considered. Typed functional programs use variables of any order and constants of order $\leq 1$, where constants of order 1 are strongly computable, monotonic functions with indeterminate values of arguments. The basic semantics of the typed functional program is a function with indeterminate values of arguments, which is the main component of its least solution. The interpretation algorithms of typed functional programs are based on substitutions, $\beta$-reduction and canonical $\delta$-reduction. The basic semantics of the untyped functional program is the untyped $\lambda$-term, which is defined by means of the fixed point combinator. The interpretation algorithms of untyped functional programs are based on substitutions and $\beta$-reduction. Interpretation algorithms are examined for completeness and comparability. It is investigated how the "behavior" of the interpretation algorithm changes after translation of typed functional program into untyped functional program.

**MSC2010:** 68N18.

***Keywords*:** typed functional program, untyped functional program, basic semantics, interpretation algorithm, completeness, comparability, translation.

**Introduction.** The paper is devoted to typed and untyped functional programs. A typed functional program is a system of equations (with separating variables) in the monotonic models of typed $\lambda$-calculus. Typed functional programs use variables of any order and constants of order $\leq 1$, where constants of order 1 are strongly computable functions with indeterminate values of arguments. The basic semantics of the typed functional program is a function with indeterminate values of arguments, which is the main component of its least solution (see [1–5]). The interpretation algorithms of typed functional programs are based on substitutions, $\beta$-reduction and canonical $\delta$-reduction [6]. An untyped functional program is a system of equations (with separating variables) in the untyped $\lambda$-calculus. The basic semantics of the untyped functional program is the untyped $\lambda$-term, which is defined by means of the

[*] E-mail: nigiyan@ysu.am

fixed point combinator (see [5, 7, 8]). The interpretation algorithms of untyped functional programs are based on substitutions and $\beta$-reduction. In [5] an algorithm that translates the typed functional program $P$ into the untyped functional program $P'$ is suggested. It is proved that the basic semantics of the program $P'$ $\lambda$-defines the basic semantics of the program $P$. For typed and untyped functional programs, the four classical interpretation algorithms are examined for completeness and comparability. Those algorithms are the followings: FSRNF (algorithm of full substitution and reduction to the normal form); LSRNF (algorithm of left substitution and reduction to the normal form); ACT (active algorithm); PAS (passive algorithm). It is investigated how the "behavior" of these interpretation algorithms changes after the translation of typed functional program $P$ into untyped functional program $P'$.

**Interpretation of Typed Functional Programs.** The definitions of this section can be found in [1–6]. A partially ordered set is said to be complete, if each of its linear ordered subsets has the least upper bound. It is easy to see that every complete set has the least element. Let $A, B$ be nonempty partially ordered sets. A mapping $\varphi : A \to B$ is said to be monotonic, if $a \sqsubseteq b$ implies $\varphi(a) \sqsubseteq \varphi(b)$ for all $a, b \in A$ ($\sqsubseteq$ is the symbol of partial ordering relation).

Let $M$ be a partially ordered set, which has an element $\perp$, which corresponds to the indeterminate value. Each element of $M$ is comparable only with itself and with $\perp$, which is the least element of $M$. Let us define the set of types (denoted by $Types$).

1. $M \in Types$;

2. If $\beta, \alpha_1, \ldots, \alpha_k \in Types$ $(k > 0)$, then the set of all monotonic mappings from $\alpha_1 \times \ldots \times \alpha_k$ into $\beta$ $\big($denoted by $[\alpha_1 \times \ldots \times \alpha_k \to \beta]\big)$ belongs to $Types$.

Let $\alpha \in Types$, then the order of type $\alpha$ $\big($denoted by $ord(\alpha)\big)$ will be a natural number, which is defined in the following way: if $\alpha = M$, then $ord(\alpha) = 0$, if $\alpha = [\alpha_1 \times \ldots \times \alpha_k \to \beta]$, where $\beta, \alpha_1, \ldots, \alpha_k \in Types$, $k > 0$, then $ord(\alpha) = 1 + \max(ord(\alpha_1), \ldots, ord(\alpha_k), ord(\beta))$. If $x$ is a variable of type $\alpha$ and a constant $c \in \alpha$, then $ord(x) = ord(c) = ord(\alpha)$. Every type $\alpha \in Types$ is a complete set (see [1]).

Let $\alpha \in Types$ and $V^T_\alpha$ be a countable set of variables of type $\alpha$, then $V^T = \bigcup\limits_{\alpha \in Types} V^T_\alpha$ is the set of all variables. The set of all terms, denoted by $\Lambda^T = \bigcup\limits_{\alpha \in Types} \Lambda^T_\alpha$, where $\Lambda^T_\alpha$ is the set of terms of type $\alpha$, is defined by:

1. If $c \in \alpha$, $\alpha \in Types$, then $c \in \Lambda^T_\alpha$;

2. If $x \in V^T_\alpha, \alpha \in Types$, then $x \in \Lambda^T_\alpha$;

3. If $\tau \in \Lambda^T_{[\alpha_1 \times \ldots \times \alpha_k \to \beta]}$, $t_i \in \Lambda^T_{\alpha_i}$, where $\beta, \alpha_i \in Types$, $i = 1, \ldots, k$, $k \geq 1$, then $\tau(t_1, \ldots, t_k) \in \Lambda^T_\beta$ (the operation of application);

4. If $\tau \in \Lambda^T_\beta, x_i \in V^T_{\alpha_i}$, where $\beta, \alpha_i \in Types$, $i \neq j \Rightarrow x_i \neq x_j$, $i, j = 1, \ldots, k$, $k \geq 1$, then $\lambda x_1 \ldots x_k[\tau] \in \Lambda^T_{[\alpha_1 \times \ldots \times \alpha_k \to \beta]}$ (the operation of abstraction).

The notions of free and bound occurrences of variables in terms as well as the notion of a free variable are introduced in the conventional way. The set of all free variables of a term $t$ is denoted by $FV(t)$. A term, which does not contain free variables is called a closed term. Terms $t_1$ and $t_2$ are said to be congruent (which is denoted by $t_1 \equiv t_2$), if one term can be obtained from the other by renaming bound variables. In what follows, congruent terms are considered identical.

Let $t \in \Lambda^T_\alpha$, $\alpha \in Types$ and $FV(t) \subset \{y_1,\dots,y_n\}$, $\bar{y_0} = \langle y^0_1,\dots,y^0_n\rangle$, where $y_i \in V^T_{\beta_i}$, $y^0_i \in \beta_i$, $\beta_i \in Types$, $i = 1,\dots,n$, $n \geq 0$. The value of the term $t$ for the values of the variables $y_1,\dots,y_n$ equal to $\bar{y_0} = \langle y^0_1,\dots,y^0_n\rangle$ is denoted by $Val_{\bar{y_0}}(t)$ and is defined in the conventional way (see [1]). It follows from [1], that for any $\bar{y_0} =< y^0_1,\dots,y^0_n >$ and $\bar{y_1} =< y^1_1,\dots,y^1_n >$ such that $\bar{y_0} \sqsubseteq \bar{y_1}$, where $y^0_i, y^1_i \in \beta_i$ $(1 \leq i \leq n)$, we have the following: $Val_{\bar{y_0}}(t) \in \alpha$ and $Val_{\bar{y_0}}(t) \sqsubseteq Val_{\bar{y_1}}(t)$.

Let terms $t_1,t_2 \in \Lambda^T_\alpha$, $\alpha \in Types$, $FV(t_1) \cup FV(t_2) = \{y_1,\dots,y_n\}$, $y_i \in V^T_{\beta_i}$, $\beta_i \in Types$, $i = 1,\dots,n$, $n \geq 0$, then terms $t_1$ and $t_2$ are called equivalent (denoted by $t_1 \sim t_2$), if for any $\bar{y_0} = \langle y^0_1,\dots,y^0_n\rangle$, where $y^0_i \in \beta_i$, $i = 1,\dots,n$, we have: $Val_{\bar{y_0}}(t_1) = Val_{\bar{y_0}}(t_2)$. A term $t \in \Lambda^T_\alpha, \alpha \in Types$, is called a constant term with value $a \in \alpha$ if $t \sim a$.

To show mutually different variables of interest $x_1,\dots,x_k, k \geq 1$, of a term $t$, the notation $t[x_1,\dots,x_k]$ is used. The notation $t[t_1,\dots,t_k]$ denotes the term obtained by the simultaneous substitution of the terms $t_1,\dots,t_k$ for all free occurrences of the variables $x_1,\dots,x_k$ respectively, where $x_i \in V^T_{\alpha_i}$, $i \neq j \Rightarrow x_i \not\equiv x_j$, $t_i \in \Lambda^T_{\alpha_i}$, $\alpha_i \in Types$, $i,j = 1,\dots,k$, $k \geq 1$. A substitution is said to be admissible, if all free variables of the term being substituted remain free after the substitution. We will consider only admissible substitutions.

A term $t \in \Lambda^T$ with a fixed occurrence of a subterm $\tau_1 \in \Lambda^T_\alpha$, where $\alpha \in Types$, is denoted by $t_{\tau_1}$, and a term with this occurrence of $\tau_1$ replaced by $\tau_2$, where $\tau_2 \in \Lambda^T_\alpha$, is denoted by $t_{\tau_2}$.

Further, we assume that $M$ is a recursive set and considered terms use variables of any order and constants of order $\leq 1$, where constants of order 1 are strongly computable, monotonic functions with indeterminate values of arguments. A function $f : M^k \to M, k \geq 1$, with indeterminate values of arguments is said to be strongly computable, if there exists an algorithm, which stops with value $f(m_1,...,m_k)$ for all $m_1,\dots,m_k \in M$ (see [3,4]). Suppose that each strongly computable function with indeterminate values of arguments is given by its algorithm. Hereafter, all such terms will be denoted by $\Lambda^T$ and all such terms of type $\alpha$ will be denoted by $\Lambda^T_\alpha$.

A term of the form $\lambda x_1 \dots x_k[\tau[x_1,\dots,x_k]](t_1,\dots,t_k)$, where $x_i \in V^T_{\alpha_i}$, $i \neq j \Rightarrow x_i \not\equiv x_j$, $\tau \in \Lambda^T$, $t_i \in \Lambda^T_{\alpha_i}$, $\alpha_i \in Types$, $i,j = 1,\dots,k$, $k \geq 1$, is called a $\beta$-redex, its convolution is the term $\tau[t_1,\dots,t_k]$. A one-step $\beta$-reduction ($\to_\beta$) and $\beta$-reduction ($\to\!\!\to_\beta$) are defined in the conventional way (see [5, 6]). A term containing no $\beta$-redexes is called a $\beta$-normal form. The set of all $\beta$-normal forms is denoted by $\beta - NF^T$.

$\delta$-redex has a form $f(t_1,\dots,t_k)$, where $f \in [M^k \to M]$, $t_i \in \Lambda^T_M$, $i = 1,\dots,k$, $k \geq 1$, its convolution is either $m \in M$ and in this case $f(t_1,\dots,t_k) \sim m$ or a subterm $t_i$ and in this case $f(t_1,\dots,t_k) \sim t_i, 1 \leq i \leq k$. A one-step $\delta$-reduction ($\to_\delta$) and $\delta$-reduction ($\to\!\!\to_\delta$) are defined in the conventional way (see [5, 6]). A term containing no $\delta$-redexes is called a $\delta$-normal form.

A one-step $\beta\delta$-reduction ($\to_{\beta\delta}$) and $\beta\delta$-reduction($\to\!\!\to_{\beta\delta}$) are defined in the conventional way (see [5, 6]). It follows from [9], that if $t_1 \to\!\!\to_{\beta\delta} t_2$, then $t_1 \sim t_2$, where $t_1, t_2 \in \Lambda^T_\alpha$, $\alpha \in Types$. A term containing no $\beta\delta$-redexes is called a normal form. The set of all normal forms is denoted by $NF^T$. For every term $t \in \Lambda$ there exists a term $\tau \in NF^T$ such that $t \to\!\!\to_{\beta\delta} \tau$ (see [9]).

A notion of $\delta$-reduction is called a single-valued notion of $\delta$-reduction, if $\delta$ is a single-valued relation, i.e. if $\langle\tau_0, \tau_1\rangle \in \delta$ and $\langle\tau_0, \tau_2\rangle \in \delta$, then $\tau_1 \equiv \tau_2$, where $\tau_0, \tau_1, \tau_2 \in \Lambda^T_M$.

A notion of $\delta$-reduction is called an effective notion of $\delta$-reduction, if there exists an algorithm, which for any term $f(t_1, \dots, t_k)$, where $f \in [M^k \to M]$, $t_i \in \Lambda^T_M$, $i = 1, \dots, k$, $k \geq 1$, gives its convolution if $f(t_1, \dots, t_k)$ is a $\delta$-redex and stops with a negative answer otherwise.

***Definition 1*** [6]. An effective, single-valued notion of $\delta$-reduction is called a canonical notion of $\delta$-reduction, if:

1. $t \in \beta - NF^T,\ t \sim m,\ m \in M \setminus \{\perp\} \Rightarrow t \to\!\!\to_\delta m$;

2. $t \in \beta - NF^T,\ FV(t) = \emptyset,\ t \sim \perp \Rightarrow t \to\!\!\to_\delta \perp$.

In [6] it was proved, that for every recursive set of strongly computable, monotonic functions with indeterminate values of arguments, there exists a canonical notion of $\delta$-reduction. Further, we will only use the canonical notion of $\delta$-reduction.

Typed functional program $P$ is the following system of equations:

$$\begin{aligned} F_1 &= t_1[F_1, \dots, F_n], \\ &\dots \\ F_n &= t_n[F_1, \dots, F_n], \end{aligned} \tag{1}$$

where $F_i \in V^T_{\alpha_i}$, $i \neq j \Rightarrow F_i \not\equiv F_j$, $t_i[F_1, ..., F_n] \in \Lambda^T_{\alpha_i}$, $FV(t_i[F_1, \dots, .F_n]) \subset \{F_1, \dots, F_n\}$, $\alpha_i \in Types$, $i, j = 1, \dots, n$, $n \geq 1$, $\alpha_1 = [M^k \to M]$, $k \geq 1$.

We consider the mapping $\Psi_P : \alpha_1 \times \ldots \times \alpha_n \to \alpha_1 \times \ldots \times \alpha_n$, which is defined as follows: if $\bar{g} = \langle g_1, \dots, g_n\rangle$, where $g_i \in \alpha_i$, $i = 1, \dots, n$, then $\Psi_P(\bar{g}) = \langle Val_{\bar{g}}(t_1[F_1, \dots, F_n]), \dots, Val_{\bar{g}}(t_n[F_1, \dots, F_n])\rangle$. $\bar{g}$ is said to be a solution of the program $P$, if $\Psi_P(\bar{g}) = \bar{g}$, i.e. $\langle Val_{\bar{g}}(t_1[F_1, \dots, F_n]), \dots, Val_{\bar{g}}(t_n[F_1, ..., F_n])\rangle = \langle g_1, ..., g_n\rangle$. Every typed functional program $P$ has a least solution (see [1]). Let $\langle f_1, \dots, f_n\rangle \in \alpha_1 \times \ldots \times \alpha_n$ be the least solution of $P$, then the first component $f_1 \in [M^k \to M]$ of the least solution is said to be the basic semantics of the program $P$ and is denoted by $f_P$. $Fix(P) = \{(m_1, \dots, m_k, m) | f_P(m_1, \dots, m_k) = m$, where $m, m_1, \dots, m_k \in M, k \geq 1\}$.

***Theorem 1*** (on basic semantics of typed functional programs). Let $f_P \in [M^k \to M]$, $k \geq 1$, be the basic semantics of a typed functional program $P$ of the form (1), then for all $m_1, \dots, m_k \in M$ we have:

$$f_P(m_1, \dots m_k) = sup\{\Psi^s_P(\bar{\Omega})_1(m_1, \dots, m_k) \mid s < \omega\},$$

where $\bar{\Omega}$ is the least element of the set $\alpha_1 \times \ldots \times \alpha_n$, $\Psi^0_P(\bar{\Omega}) = \bar{\Omega}$, $\Psi^{s+1}_P(\bar{\Omega}) = \Psi_P(\Psi^s_P(\bar{\Omega}))$ and $\Psi^s_P(\bar{\Omega})_1$ is the first component of the $\Psi^s_P(\bar{\Omega})$, $s < \omega$, $\omega$ is the ordinal corresponding to the set of natural numbers.

***Proof.*** Proof follows from [2]. $\square$

***Theorem 2*** (on substitutions for typed functional programs) [5]. Let $f_P \in [M^k \to M]$, $k \geq 1$, be the basic semantics of a typed functional program $P$ of the form (1), let $\delta$ be a canonical notion of $\delta$-reduction, then for all $m_1,\dots,m_k \in M$ we have: $f_P(m_1,\dots,m_k) = m \neq \perp \Leftrightarrow \exists s \geq 1, t_1^s[F_1,\dots,F_n](m_1,\dots,m_k) \twoheadrightarrow_{\beta\delta} m \neq \perp$, where $t_i^0[F_1,\dots,F_n] \equiv F_i$, $t_i^r[F_1,\dots,F_n] \equiv t_i[t_1^{r-1}[F_1,\dots,F_n],\dots,t_n^{r-1}[F_1,\dots,F_n]]$, $r \geq 1$, $i = 1,\dots,n$, $n \geq 1$.

***Interpretation Algorithms.*** The input of the interpretation algorithm A is a program $P$ of the form (1), a term $t \in \Lambda^T$, and a canonical notion of $\delta$-reduction. Algorithm A stops with result $A(P,t) \in NF^T$, $FV(A(P,t)) \cap \{F_1,\dots,F_n\} = \emptyset$ or works infinitely. Algorithm A uses three kinds of operations:

1. substitution of the terms $t_1,\dots t_n$ instead of some free occurrences of the variables $F_1,\dots,F_n$;

2. one-step $\beta$-reduction;

3. one-step $\delta$-reduction.

$Proc_A(P) = \{(m_1,\dots,m_k,m)|$ algorithm A stops for program $P$ and term $F_1(m_1,\dots,m_k)$ with a result $m$, where $m,m_1,\dots,m_k \in M, k \geq 1\}$.

Interpretation algorithm A is consistent, if for any program $P$ and for any canonical notion of $\delta$-reduction we have: $Proc_A(P) \subset Fix(P)$.

***Theorem 3.*** Every interpretation algorithm is consistent.

***Proof.*** Proof follows from the results of [10]. □

Interpretation algorithm A is complete, if for any program $P$ of the form (1) and for any canonical notion of $\delta$-reduction we have: if $(m_1,\dots,m_k,m) \in Fix(P)$ and $m \neq \perp$, then $(m_1,\dots,m_k,m) \in Proc_A(P)$, $m,m_1,\dots,m_k \in M, k \geq 1$.

Let A and B be interpretation algorithms, then A<B, if for any program $P$ and any canonical notion of $\delta$-reduction we have: if $(m_1,\dots,m_k,m) \in Proc_A(P)$ and $m \neq \perp$, then $(m_1,\dots,m_k,m) \in Proc_B(P)$, $m,m_1,\dots,m_k \in M$, $k \geq 1$.

Interpretation algorithms A and B are incomparable, if A$\not<$ B and B$\not<$A.

Interpretation algorithm A depends on canonical notion of $\delta$-reduction, if there exist a program $P$, $m_1,\dots,m_k \in M$, $k \geq 1$, and canonical notions of $\delta$-reduction $\delta_1$ and $\delta_2$ such, that A stops on $P$ and $F_1(m_1,\dots,m_k)$ with a result $m \in M, m \neq \perp$, for $\delta_1$ and works infinitely for $\delta_2$.

We explore four classical interpretation algorithms: FSRNF (algorithm of full substitution and reduction to the normal form); LSRNF (algorithm of left substitution and reduction to the normal form); ACT (active algorithm); PAS (passive algorithm).

Algorithm FSRNF.

1. If $t \in NF^T$ and $FV(t) \cap \{F_1,\dots,F_n\} = \emptyset$, then $t$ else go to 2.

2. If $t \notin NF^T$ and $t \equiv t_\tau$, where $\tau$ is leftmost redex ($\beta$-redex or $\delta$-redex), then FSRNF($P,t_{\tau'}$), where $\tau'$ is the convolution of the redex $\tau$, else go to 3.

3. If $t \equiv t[F_1,\dots,F_n]$, then FSRNF $(P,t[t_1,\dots,t_n])$.

Algorithm LSRNF.

1. If $t \in NF^T$ and $FV(t) \cap \{F_1,\dots,F_n\} = \emptyset$, then $t$ else go to 2.

2. If $t \notin NF^T$ and $t \equiv t_\tau$, where $\tau$ is leftmost redex ($\beta$-redex or $\delta$-redex), then LSRNF $(P, t_{\tau'})$, where $\tau'$ is the convolution of the redex $\tau$, else go to 3.

3. If $t \equiv t_{F_i}(1 \le i \le n)$, where $t_{F_i}$ is the term $t$ with a fixed leftmost free occurrence of a variable of the set $\{F_1, \dots, F_n\}$, then LSNFR$(P, t_{t_i})$.

Algorithm ACT.

1. If $t \in NF^T$ and $FV(t) \cap \{F_1, \dots, F_n\} = \emptyset$, then $t$, else go to 2.

2. If $t \equiv t_{F_i}(1 \le i \le n)$, where $t_{F_i}$ is the term $t$ with a fixed leftmost free occurrence of a variable of the set $\{F_1, \dots, F_n\}$, which is located to the left of the leftmost redex, then ACT$(P, t_{t_i})$ else go to 3.

3. If $t \equiv t_{\lambda x_1 \dots x_r[\tau[x_1,\dots,x_r]](\tau_1,\dots,\tau_r)}$, where $\lambda x_1 \dots x_r[\tau[x_1, \dots, x_r]](\tau_1, \dots, \tau_r)$ is the leftmost redex, then ACT $(P, t_{\tau[\mathrm{ACT}(P,\tau_1),\dots,\mathrm{ACT}(P,\tau_r)]})$ else go to 4.

4. If $t \equiv t_\tau$, where $\tau$ is the leftmost redex, which is a $\delta$-redex, then ACT$(P, t_{\tau'})$, where $\tau'$ is the convolution of the $\delta$-redex $\tau$.

Algorithm PAS.

1. If $t \in NF^T$ and $FV(t) \cap \{F_1, \dots, F_n\} = \emptyset$, then $t$, else go to 2.

2. If $t \equiv t_{F_i}(1 \le i \le n)$, where $t_{F_i}$ is the term $t$ with a fixed leftmost free occurrence of a variable of the set $\{F_1, \dots, F_n\}$ which is located to the left of the leftmost redex, then PAS$(P, t_{t_i})$ else go to 3.

3. If $t \equiv t_{\lambda x_1 \dots x_r[\tau[x_1,\dots,x_r]](\tau_1,\dots,\tau_r)}$, where $\lambda x_1 \dots x_r[\tau[x_1, \dots, x_r]](\tau_1, \dots, \tau_r)$ is the leftmost redex, then PAS$(P, t_{\tau[\tau_1,\dots,\tau_r]})$ else go to 4.

4. If $t \equiv t_\tau$, where $\tau$ is the leftmost redex, which is a $\delta$-redex, then PAS$(P, t_{\tau'})$, where $\tau'$ is the convolution of the $\delta$-redex $\tau$.

***Theorem 4.*** Interpretation algorithm FSRNF is complete.

***Proof.*** Let $f_P(m_1, \dots, m_k) = m \neq \perp$, then according to the Theorem 2, there exists such $s \ge 1$, that $t_1^s[F_1, \dots, F_n](m_1, \dots, m_k) \twoheadrightarrow_{\beta\delta} m$, therefore, according to [9], $t_1^s[F_1, \dots, F_n](m_1, \dots, m_k) \sim m$. Let, the following sequence of normal forms corresponds to FSRNF algorithm: $t_1'[F_1, \dots, F_n]$, $t_2'[F_1, \dots, F_n], \dots, t_s'[F_1, \dots, F_n]$. Let us show, that for every $r = 1, \dots, s$, $t_1^r[F_1, \dots, F_n](m_1, \dots, m_k) \sim t_r'[F_1, \dots, F_n]$.
$t_1[F_1, \dots, F_n](m_1, \dots, m_k) \twoheadrightarrow_{\beta\delta} t_1'[F_1, \dots, F_n]$ and, according to [9],
$t_1[F_1, \dots, F_n](m_1, \dots, m_k) \sim t_1'[F_1, \dots, F_n]$, and
$t_1^2[F_1, \dots, F_n](m_1, \dots, m_k) \sim t_1'[t_1[F_1, ..., F_n], ..., t_n[F_1, ..., F_n]]$,
$t_1'[t_1[F_1, ..., F_n], \dots, t_n[F_1, \dots, F_n]] \twoheadrightarrow_{\beta\delta} t_2'[F_1, \dots, F_n]$ and, according to [9],
$t_1^2[F_1, \dots, F_n](m_1, \dots, m_k) \sim t_2'[F_1, \dots, F_n]$, and
$t_1^3[F_1, \dots, F_n](m_1, \dots, m_k) \sim t_2'[t_1[F_1, \dots, F_n], \dots, t_n[F_1, \dots, F_n]]$,
$t_2'[t_1[F_1, \dots, F_n], \dots, t_n[F_1, \dots, F_n]] \twoheadrightarrow_{\beta\delta} t_3'[F_1, \dots, F_n]$ and, according to [9],
$t_1^3[F_1, \dots, F_n](m_1, \dots, m_k) \sim t_3'[F_1, \dots, F_n]$, and so on.

Since $t_1^s[F_1, ..., F_n](m_1, ..., m_k) \sim t_s'[F_1, ..., F_n]$ and $t_1^s[F_1, ..., F_n](m_1, ..., m_k) \sim m$, we have $t_s'[F_1, ..., F_n] \sim m$. Since $t_s'[F_1, ..., F_n] \in NF^T$, we have $t_s'[F_1, ..., F_n] \in \beta - NF^T$ and, according to point 1 of Definition 1, $t_s'[F_1, ..., F_n] \twoheadrightarrow_\delta m$, but $t_s'[F_1, ..., F_n] \in NF^T$, therefore $t_s'[F_1, ..., F_n] \equiv m$. □

Let us fix the set $M = N \cup \{\perp\}$, where $N = \{0,1,2,\ldots\}$, built-in functions $sg1, sg2\ \ \& \in [M^2 \to M]$, and three canonical notions of $\delta$-reduction: $\delta, \delta_1, \delta_2$.

For $m_1, m_2 \in M$ we have:

$sg1(m_1,m_2)$ equals 0, if $m_1 = 0$,
equals 1, if $m_1 \neq \perp$ and $m_1 \geq 1$,
equals $\perp$, if $m_1 = \perp$.
$sg2(m_1,m_2)$ equals 0, if $m_2 = 0$,
equals 1, if $m_2 \neq \perp$ and $m_2 \geq 1$,
equals $\perp$, if $m_2 = \perp$.
$\&(m_1,m_2)$ equals 0, if $m_1 = 0$ or $m_2 = 0$,
equals 1, if $m_1 \neq \perp, m_2 \neq \perp$ and $m_1 \geq 1, m_2 \geq 1$,
equals $\perp$, otherwise.

It is easy to see, that $sg1, sg2\ \ \&$ are strongly computable, monotonic functions with indeterminate values of arguments; $sg1, sg2$ are $\lambda$-definable functions, & is not $\lambda$-definable function (see [3,4]).

Canonical notion of $\delta$-reduction $\delta$:
$\langle sg1(0,t), 0\rangle \in \delta$, where $t \in \Lambda^T_M$,
$\langle sg1(n,t), 1\rangle \in \delta$, where $n \in N$, $n \geq 1$, $t \in \Lambda^T_M$,
$\langle sg1(\perp,t), \perp\rangle \in \delta$, where $t \in \Lambda^T_M$,
$\langle sg2(t,0), 0\rangle \in \delta$, where $t \in \Lambda^T_M$,
$\langle sg2(t,n), 1\rangle \in \delta$, where $n \in N$, $n \geq 1$, $t \in \Lambda^T_M$,
$\langle sg2(t,\perp), \perp\rangle \in \delta$, where $t \in \Lambda^T_M$,
$\langle \&(0,t), 0\rangle \in \delta$, where $t \in \Lambda^T_M$,
$\langle \&(t,0), 0\rangle \in \delta$, where $t \in \Lambda^T_M$,
$\langle \&(n_1,n_2), 1\rangle \in \delta$, where $n_1$, $n_2 \in N$, $n_1, n_2 \geq 1$,
$\langle \&(\perp,n), \perp\rangle \in \delta$, where $n \in N$, $n \geq 1$,
$\langle \&(n,\perp), \perp\rangle \in \delta$, where $n \in N$, $n \geq 1$,
$\langle \&(\perp,\perp), \perp\rangle \in \delta$.

Canonical notion of $\delta$-reduction $\delta_1$:
$\delta_1 = \delta \cup \{\langle \&(\&(t_1,t_2), \&(t_2,t_1)), \&(t_1,t_2)\rangle \mid t_1, t_2 \in \Lambda^T_M\}$.

Canonical notion of $\delta$-reduction $\delta_2$:
$\delta_2 = \delta \cup \{\langle \&(\&(t_1,t_2), \&(t_2,t_1)), \&(t_2,t_1)\rangle \mid t_1, t_2 \in \Lambda^T_M\}$.

***Theorem 5.*** Interpretation algorithms LSRNF, ACT, PAS are not complete.

***Proof.*** Incompleteness of the algorithm ACT is obvious, it already exists for programs that do not use built-in functions. Incompleteness of the algorithm LSRNF follows from the results of [11]. Incompleteness of the algorithm PAS follows from the results of [10]. In [10] and [11] incompleteness of LSRNF and PAS were proved for the programs that use not $\lambda$-definable functions. We will show incompleteness of the algorithms LSRNF, ACT, PAS for the programs that can use only $\lambda$-definable functions.

Let $M = N \cup \{\perp\}$, $x \in V^T_M$, $F_1, F_2, F_3, F_4 \in V^T_{[M\to M]}$.

Program $P_1$ is: $F_1 = \lambda x[F_2(F_3(x))]$,

$F_2 = \lambda x[0]$,
$F_3 = \lambda x[F_3(x)]$.

For all $m \in Mf_{P_1}(m) = 0$, therefore, $(1,0) \in Fix(P_1)$. Let us show, that $(1,0) \notin Proc_{\text{ACT}}(P_1)$. For ACT we have: $F_1(1); \lambda x[F_2(F_3(x))](1) \to_\beta F_2(F_3(1))$; $\lambda x[0](F_3(1))$; now we must apply algorithm ACT to the term $F_3(1); \lambda x[F_3(x)](1) \to_\beta F_3(1); \ldots$ and so on.

Program $P_2$ is: $F_1 = \lambda x[F_2(F_3(x))]$,
$F_2 = \lambda x[sg2(F_4(x), x)]$,
$F_3 = \lambda x[0]$,
$F_4 = \lambda x[F_4(x)]$.

For all $m \in Mf_{P_2}(m) = 0$, therefore, $(1,0) \in Fix(P_2)$. Let us show, that $(1,0) \notin Proc_{\text{LSRNF}}(P_2)$ and $(1,0) \notin Proc_{\text{PAS}}(P_2)$. For LSNFR and PAS we have: $F_1(1); \lambda x[F_2(F_3(x))](1) \to_\beta F_2(F_3(1)); \lambda x[sg2(F_4(x), x)](F_3(1)) \to_\beta sg2(F_4(F_3(1)), F_3(1))$; $sg2(\lambda x[F_4(x)](F_3(1)), F_3(1)) \to_\beta sg2(F_4(F_3(1)), F_3(1)); \ldots$ and so on. $\square$

***Theorem 6.*** Interpretation algorithms LSRNF, ACT, PAS are pairwise incomparable.

***Proof.*** Incomparability of algorithms ACT and PAS, ACT and LSRNF follows from [10]. In [10] this has been proved for programs that use not $\lambda$-definable functions. We will show incomparability of algorithms ACT and PAS, ACT and LSRNF for programs that can use only $\lambda$-definable functions.

Let us show, that $(1,0) \notin Proc_{\text{ACT}}(P_1), (1,0) \in Proc_{\text{PAS}}(P_1)$ and $(1,0) \in Proc_{\text{LSRNF}}(P_1)$. From the proof of Theorem 5 it follows that $(1,0) \notin Proc_{\text{ACT}}(P_1)$. Let us show, that $(1,0) \in Proc_{\text{PAS}}(P_1)$ and $(1,0) \in Proc_{\text{LSRNF}}(P_1)$. For PAS and LSRNF we have: $F_1(1)$; $\lambda x[F_2(F_3(x))](1) \to_\beta F_2(F_3(1))$; $\lambda x[0](F_3(1)) \to_\beta 0$.

Let us show, that $(1,0) \in Proc_{\text{ACT}}(P_2), (1,0) \notin Proc_{\text{PAS}}(P_2)$ and $(1,0) \notin Proc_{\text{LSRNF}}(P_2)$. From the proof of Theorem 5 it follows that $(1,0) \notin Proc_{\text{PAS}}(P_2)$ and $(1,0) \notin Proc_{\text{LSRNF}}(P_2)$. Let us show, that $(1,0) \in Proc_{\text{ACT}}(P_2)$. For ACT we have: $F_1(1); \lambda x[F_2(F_3(x))](1) \to_\beta F_2(F_3(1)); \lambda x[sg2(F_4(x), x)](F_3(1))$; now we must apply algorithm ACT to the term $F_3(1); \lambda x[0](1) \to_\beta 0$; further we have $\lambda x[sg2(F_4(x), x)](0) \to_\beta sg2(F_4(0), 0) \to_\delta 0$.

Now we show that the algorithms PAS and LSRNF are incomparable.

Program $P_3$ is : $F_1 = \lambda x[sg2(F_2(x), sg2(1, x))]$,
$F_2 = \lambda x[F_2(x)]$.

Let us show that $(0,0) \notin Proc_{\text{PAS}}(P_3)$ and $(0,0) \in Proc_{\text{LSRNF}}(P_3)$.

For PAS we have: $F_1(0)$; $\lambda x[sg2(F_2(x), sg2(1, x))](0) \to_\beta sg2(F_2(0), sg2(1, 0))$; $sg2(\lambda x[F_2(x)](0), sg2(1, 0)) \to_\beta sg2(F_2(0), sg2(1, 0)); \ldots$ and so on.

For LSRNF we have: $F_1(0)$; $\lambda x[sg2(F_2(x), sg2(1, x))](0) \to_\beta sg2(F_2(0), sg2(1, 0)) \to_\delta sg2(F_2(0), 0) \to_\delta 0$.

Program $P_4$ is: $F_1 = \lambda x[\&(\&(F_2(x), F_3(x)), \&(F_3(x), F_2(sg1(x, 1))))]$,
$F_2 = \lambda x[0]$,
$F_3 = \lambda x[F_3(x)]$.

Let us show, that $(0,0) \in Proc_{\text{PAS}}(P_4)$ and $(0,0) \notin Proc_{\text{LSRNF}}(P_4)$.

For PAS we have: $F_1(0); \lambda x[\&(\&(F_2(x),F_3(x)),\&(F_3(x),F_2(sg1(x,1))))](0) \to_\beta$ $\&(\&(F_2(0),F_3(0)),\&(F_3(0),F_2(sg1(0,1))))$; $\&(\&(\lambda x[0](0),F_3(0)),\&(F_3(0),F_2(sg1(0,1)))) \to_\beta$ $\&(\&(0,F_3(0)),\&(F_3(0),F_2(sg1(0,1)))) \to_{\delta_2} \&(0,\&(F_3(0),F_2(sg1(0,1)))) \to_{\delta_2} 0$.

For LSRNF we have: $F_1(0); \lambda x[\&(\&(F_2(x),F_3(x)),\&(F_3(x),F_2(sg1(x,1))))](0)$ $\to_\beta \&(\&(F_2(0),F_3(0)),\&(F_3(0),F_2(sg1(0,1)))) \to_{\delta_2}$ $\&(\&(F_2(0),F_3(0)),\&(F_3(0),F_2(0))) \to_{\delta_2} \&(F_3(0),F_2(0)); \&(\lambda x[F_3(x)](0),F_2(0))$ $\to_\beta \&(F_3(0),F_2(0)); \ldots$ and so on. $\square$

***Theorem 7.*** Interpretation algorithms LSRNF, ACT, PAS depend on canonical notion of $\delta$-reduction.

***Proof.*** Let $M = N \cup \{\bot\}$, $x \in V^T_M$, $F_1,F_2,F_3 \in V^T_{[M\to M]}$.

Program $P_5$ is: $F_1 = \lambda x[\&(\&(F_2(x),F_3(x)),\&(F_3(x),F_2(x)))]$,
$F_2 = \lambda x[0]$,
$F_3 = \lambda x[F_3(x)]$.

Let us show, that $(1,0) \in Proc_{\text{LSRNF}}(P_5)$, $(1,0) \in Proc_{\text{PAS}}(P_5)$, $(1,0) \in Proc_{\text{ACT}}(P_5)$ for $\delta_1$, and $(1,0) \notin Proc_{\text{LSRNF}}(P_5), (1,0) \notin Proc_{\text{PAS}}(P_5)$, $(1,0) \notin Proc_{\text{ACT}}(P_5)$ for $\delta_2$.

Let us show for LSNFR, PAS, ACT and $\delta_1$ :
$F_1(1); \lambda x[\&(\&(F_2(x),F_3(x)),\&(F_3(x),F_2(x)))](1) \to_\beta$ $\&(\&(F_2(1),F_3(1)),\&(F_3(1),F_2(1))) \to_{\delta_1} \&(F_2(1),F_3(1)); \&(\lambda x[0](1),F_3(1)) \to_\beta$ $\&(0,F_3(1)) \to_{\delta_1} 0$.

Let us show for LSNFR, PAS, ACT and $\delta_2$ :
$F_1(1); \lambda x[\&(\&(F_2(x),F_3(x)),\&(F_3(x),F_2(x)))](1) \to_\beta$ $\&(\&(F_2(1),F_3(1)),\&(F_3(1),F_2(1))) \to_{\delta_2} \&(F_3(1),F_2(1))$; $\&(\lambda x[F_3(x)](1),F_2(1)) \to_\beta \&(F_3(1),F_2(1)); \ldots$ and so on. $\square$

**Interpretation of Untyped Functional Programs.** The definitions of this section can be found in [7, 8]. Let us fix a countable set of variables $V$. The set $\Lambda$ of terms is defined as follows:

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

The following shorthand notations are introduced: a term $(\ldots(t_1 t_2)\ldots t_k)$, where $t_i \in \Lambda$, $i = 1,\ldots,k$, $k > 1$, is denoted by $t_1 t_2 \ldots t_k$, and a term $(\lambda x_1(\lambda x_2(\ldots(\lambda x_n t)\ldots)))$, where $x_j \in V$, $j = 1,\ldots,n$, $n > 0$, $t \in \Lambda$, is denoted by $\lambda x_1 x_2 \ldots x_n.t$.

The notions of free and bound occurrences of variables in terms as well as the notion of free variable are introduced in the conventional way. The set of all free variables of a term $t$ is denoted by $FV(t)$. A term, which does not contain free variables, is called closed term. Terms $t_1$ and $t_2$ are said to be congruent (which is denoted by $t_1 \equiv t_2$), if one term can be obtained from the other by renaming bound variables. In what follows, congruent terms are considered identical.

To show mutually different variables of interest $x_1,\ldots,x_k$, $k \geq 1$, of a term $t$, the notation $t[x_1,\ldots,x_k]$ is used. The notation $t[t_1,\ldots,t_k]$ denotes the term obtained by the simultaneous substitution of the terms $t_1,\dots,t_k$ for all free occurrences of variables $x_1,\dots,x_k$ respectively, $i \neq j \Rightarrow x_i \not\equiv x_j,\ i,j = 1,\dots,k,\ k \geq 1$. A substitution is said to be admissible, if all free variables of the term being substituted remain free after the substitution. We will consider only admissible substitutions.

A term $t$ with a fixed occurrence of a subterm $\tau_1$ is denoted by $t_{\tau_1}$, and a term with this occurrence of $\tau_1$ replaced by a term $\tau_2$ is denoted by $t_{\tau_2}$.

A term of the form $(\lambda x.t[x])\tau$, where $x \in V,\ t,\tau \in \Lambda$, is called a $\beta$-redex and the term $t[\tau]$ is called its convolution. A one-step $\beta$-reduction $(\to_\beta)$, $\beta$-reduction $(\to\!\!\to_\beta)$ and $\beta$-equality $(=_\beta)$ are defined in a standard way [7]. A term containing no $\beta$-redexes is called a normal form. The set of all normal forms is denoted by $NF$ and the set of all closed normal forms is denoted by $NF^0$. A term $t$ is said to have a normal form, if there exists a term $\tau$ such that $\tau \in NF$ and $t \to\!\!\to_\beta \tau$. From the Church–Rosser theorem [7] it follows, that if $t \to\!\!\to_\beta \tau_1, t \to\!\!\to_\beta \tau_2, \tau_1, \tau_2 \in NF$, then $\tau_1 \equiv \tau_2$.

If a term has a form $\lambda x_1 \dots x_k.xt_1 \dots t_n$, where $x_1,\dots,x_k,\ x \in V,\ t_1,\dots,t_n \in \Lambda$, $k,n \geq 0$, it is called a head normal form and $x$ is called its head variable. The set of all head normal forms is denoted by $HNF$. A term $t$ is said to have a head normal form, if there exists a term $\tau$, such that $\tau \in HNF$ and $t \to\!\!\to_\beta \tau$. It is known, that $NF \subset HNF$, but $HNF \not\subset NF$ (see [7]).

A term $Z \in \Lambda$ is called a fixed point combinator, if for all terms $t \in \Lambda$, $Zt =_\beta t(Zt)$. We introduce notations for some terms: $\langle t_1,\dots,t_n\rangle \equiv \lambda x.xt_1 \dots t_n$, where $x \in V,\ t_i \in \Lambda,\ x \notin FV(t_i),\ i = 1,\dots,n,\ n \geq 1,\ U_i^n \equiv \lambda x_1 \dots x_n.x_i$, where $x_j \in V,\ k \neq j \Rightarrow x_k \not\equiv x_j,\ k,j = 1,\dots,n,\ 1 \leq i \leq n,\ n \geq 1,\ P_i^n \equiv \lambda x.xU_i^n$, where $x \in V,\ 1 \leq i \leq n,\ n \geq 1, \Omega \equiv (\lambda x.xx)(\lambda x.xx)$, where $x \in V$.

Untyped functional program $P$ is the following system of equations:

$$\begin{gathered} F_1 = t_1[F_1,\dots,F_n], \\ \dots \\ F_n = t_n[F_1,\dots,F_n], \end{gathered} \tag{2}$$

where $F_i \in V,\ i \neq j \Rightarrow F_i \not\equiv F_j,\ t_i[F_1,\dots,F_n] \in \Lambda,\ FV(t_i[F_1,\dots,F_n]) \subset \{F_1,\dots,F_n\}$, $i,j = 1,\dots,n,\ \ n \geq 1$. A sequence of terms $(\tau_1,\dots,\tau_n)$ will be a solution of program $P$, if for all $i = 1,\dots,n$ we have: $\tau_i =_\beta t_i[\tau_1,\dots,\tau_n]$.

We consider the solution $(\tau_1,\dots,\tau_n)$ of the program $P$, where
$\tau_i \equiv P_i^n(Z(\lambda x.\langle t_1[P_1^n x,\dots,P_n^n x],\dots,t_n[P_1^n x,\dots,P_n^n x]\rangle)),\ \ x \in V,\ \ Z \in \Lambda$ and is a fixed point combinator, $i = 1,\dots,n$. The term $\tau_1$ is the basic semantics of the program $P$ denoted by $\tau_P$.

$Fix(P,Z) = \{(\upsilon_1,\dots,\upsilon_k,t_0) | \tau_P \upsilon_1 \dots \upsilon_k \to\!\!\to t_0,\quad t_0 \in NF^0,\ \upsilon_j \in NF^0$ or $\upsilon_j \equiv \Omega, j = 1,\dots,k,\ \ k \geq 0\}$.

***T h e o r e m  8*** (on the invariance of the basic semantics of untyped functional programs). For any untyped functional program $P$ and for any fixed point combinators $Z_1, Z_2$ we have: $Fix(P,Z_1) = Fix(P,Z_2) = Fix(P)$.

***P r o o f .*** Proof follows from the results of [12]. □

***T h e o r e m  9*** (on substitutions for untyped functional programs). Let $\tau_P$ be the basic semantics of an untyped functional program $P$ of the form (2), $\upsilon_1, \upsilon_2, \dots, \upsilon_k$

be terms, where $\upsilon_j \in NF^0$ or $\upsilon_j \equiv \Omega$, $j = 1,\dots,k$, $k \geq 0$, and $t_0 \in NF^0$, then: $\tau_P \upsilon_1 \dots \upsilon_k \to\!\!\to_\beta t_0 \Leftrightarrow \exists s \geq 1,\ t_1^s[F_1,\dots,F_n]\upsilon_1\upsilon_2\dots\upsilon_k \to\!\!\to_\beta t_0$,
where $t_i^0[F_1,\dots,F_n] \equiv F_i$, $t_i^r[F_1,\dots,F_n] \equiv t_i[t_1^{r-1}[F_1,\dots,F_n],\dots,t_n^{r-1}[F_1,\dots,F_n]]$, $r \geq 1$, $i = 1,\dots,n$, $n \geq 1$.

***Proof.*** Proof follows from the results of [8] and [12]. $\square$

***Interpretation Algorithms.*** The input of the interpretation algorithm A is a program $P$ of the form (2) and $t \in \Lambda$. Algorithm A stops with result $\mathrm{A}(P,t) \in NF, FV(\mathrm{A}(P,t)) \cap \{F_1,\dots,F_n\} = \emptyset$ or works infinitely. Algorithm A uses two kinds of operations:

1. substitution of the terms $t_1,\dots,t_n$ instead of some free occurrences of variables $F_1,\dots,F_n$;

2. one-step $\beta$-reduction.

$Proc_{\mathrm{A}}(P) = \{(\upsilon_1,\dots,\upsilon_k,t_0)|$ algorithm A stops for program $P$ and the term $F_1\upsilon_1\dots\upsilon_k$ with result $t_0 \in NF^0$, where $\upsilon_j \in NF^0$ or $\upsilon_j \equiv \Omega$, $j = 1,\dots,k$, $k \geq 0\}$.

Interpretation algorithm A is consistent, if for any program $P$ we have: $Proc_{\mathrm{A}}(P) \subset Fix(P)$.

***Theorem 10.*** Every interpretation algorithm is consistent.

***Proof.*** Proof follows from [8]. $\square$

Algorithm A is complete, if for any program $P$ we have: $Proc_{\mathrm{A}}(P) = Fix(P)$.

Let A and B be interpretation algorithms, then A<B, if for any program $P$, we have $Proc_{\mathrm{A}}(P) \subset Proc_{\mathrm{B}}(P)$.

Interpretation algorithm A and B are incomparable, if A $\not<$ B and B$\not<$ A.

For untyped functional programs, we explore the same four classical interpretation algorithms: FSRNF (algorithm of full substitution and reduction to the normal form); LSRNF (algorithm of left substitution and reduction to the normal form); ACT (active algorithm); PAS (passive algorithm).

Algorithm FSRNF.

1. If $t \in NF$ and $FV(t) \cap \{F_1,\dots,F_n\} = \emptyset$, then $t$ else go to 2.

2. If $t \notin NF$ and $t \equiv t_\tau$, where $\tau$ is left $\beta$-redex, then FSRNF$(P,t_{\tau'})$, where $\tau'$ is the convolution of the $\beta$-redex $\tau$ else go to 3.

3. If $t \equiv t[F_1,\dots,F_n]$, then FSRNF$(P,t[t_1,\dots,t_n])$.

Algorithm LSRNF.

1. If $t \in NF$ and $FV(t) \cap \{F_1,\dots,F_n\} = \emptyset$, then $t$ else go to 2.

2. If $t \notin NF$ and $t \equiv t_\tau$, where $\tau$ is left $\beta$-redex, then LSRNF$(P,t_{\tau'})$, where $\tau'$ is the convolution of the $\beta$-redex $\tau$ else go to 3.

3. If $t \equiv t_{F_i} (1 \leq i \leq n)$, where $t_{F_i}$ is the term $t$ with a fixed leftmost free occurrence of a variable of the set $\{F_1,\dots,F_n\}$, then LSRNF$(P,t_{t_i})$.

Algorithm ACT.

1. If $t \in NF$ and $FV(t) \cap \{F_1,\dots,F_n\} = \emptyset$, then $t$ else go to 2.

2. If $t \equiv t_{F_i}$ $(1 \leq i \leq n)$, where $t_{F_i}$ is the term $t$ with a fixed leftmost free occurrence of a variable of the set $\{F_1,\dots,F_n\}$, which is located to the left of the

left $\beta$-redex, then $\mathrm{ACT}(P, t_{t_i})$ else go to 3.

3. If $t \equiv t_{(\lambda x.\tau[x])\tau_1}$, where $(\lambda x.\tau[x])\tau_1$ is the left $\beta$-redex, then $\mathrm{ACT}(P, t_{\tau[\mathrm{ACT}(P,\tau_1)]})$.

Algorithm PAS.

1. If $t \in NF$ and $FV(t) \cap \{F_1, \dots, F_n\} = \emptyset$, then $t$ else go to 2.

2. If $t \equiv t_{F_i} (1 \le i \le n)$, where $t_{F_i}$ is the term $t$ with a fixed leftmost free occurrence of a variable of the set $\{F_1, \dots, F_n\}$, which is located to the left of the left $\beta$-redex, then $\mathrm{PAS}(P, t_{t_i})$ else go to 3.

3. If $t \equiv t_{(\lambda x.\tau[x])\tau_1}$, where $(\lambda x.\tau[x])\tau_1$ is the left $\beta$-redex, then $\mathrm{PAS}(P, t_{\tau[\tau_1]})$.

From the results of [8] the following three theorems follow.

***Theorem 11.*** Interpretation algorithm PAS is complete.

***Theorem 12.*** Interpretation algorithms FSRNF, LSRNF, ACT are not complete.

***Theorem 13.*** Interpretation algorithms FSRNF, LSRNF, ACT are pairwise incomparable.

**Translation and Interpretation.** Let $M$ be a recursive, partially ordered set, which has a least element $\perp$ and every element of $M$ is comparable only with itself and with $\perp$. Every $m \in M$ is mapped into an untyped term $m'$ in the following way:

If $m \in M \setminus \{\perp\}$, then $m' \in NF^0$ and for any $m_1, m_2 \in M \setminus \{\perp\}$, $m_1 \ne m_2 \Rightarrow {m_1}' \not\equiv {m_2}'$.

If $m \equiv \perp$, then $m' \equiv \Omega \equiv (\lambda x.xx)(\lambda x.xx)$.

We say that an untyped term $\Phi \in \Lambda$ $\lambda$-defines (see [3, 4]) the function $f : M^k \to M$ $(k \ge 1)$ with indeterminate values of arguments, if for all $m_1, \dots, m_k \in M$ we have:

$f(m_1, \dots, m_k) = m \ne \perp \Rightarrow \Phi {m_1}' \dots {m_k}' \twoheadrightarrow_\beta m'$,

$f(m_1, \dots, m_k) = \perp \Rightarrow \Phi {m_1}' \dots {m_k}'$ does not have a head normal form.

We consider typed terms using the functions from a recursive set $C$, every $f \in C$ is a strongly computable function with indeterminate values of arguments, which has an untyped term that $\lambda$-defines it. From [3] it follows that every $f \in C$ is a strongly computable, monotonic function with indeterminate values of arguments. Therefore, according to [6], there exists a canonical notion of $\delta$-reduction for the set $C$. Let us consider the algorithm of translation of any typed term $t$ into the untyped term $t'$ studied in [6]:

if $t \equiv m \in M$, then $t' \equiv m'$;

if $t \in C$, then $FV(t') = \emptyset$ and $t'$ $\lambda$-defines $t$;

if $t \equiv x \in V^T$, then $x' \in V$ and for any $x_1, x_2 \in V^T$, $x_1 \not\equiv x_2 \Rightarrow {x_1}' \not\equiv {x_2}'$;

if $t \equiv \tau(t_1, \dots, t_k)$, $k \ge 1$, then $t' \equiv \tau' {t_1}' \dots {t_k}'$;

if $t \equiv \lambda x_1 \dots x_n[\tau]$, $n \ge 1$, then $t' \equiv \lambda {x_1}' \dots {x_n}'.\tau'$.

Let $P$ be a typed functional program of the form (1) and $P'$ be the untyped functional program (the result of the translation) obtained by replacing typed terms for corresponding untyped terms in program $P$. Program $P'$:

$$\begin{gathered} F_1' = t_1'[F_1', \dots, F_n'], \\ \dots \\ F_n' = t_n'[F_1', \dots, F_n']. \end{gathered} \tag{3}$$

***T h e o r e m 14*** (on translation) [5]. Let $f_P \in [M^k \to M]$, $k \geq 1$, be the basic semantics of a typed functional program $P$ of the form (1) and let $\tau_{P'}$ be the basic semantics of the untyped functional program $P'$ of the form (3), then for all $m_1, \dots, m_k \in M$ we have:

$f_P(m_1, \dots, m_k) = m \neq \perp \Rightarrow \tau_{P'} m_1' \dots m_k' \twoheadrightarrow_\beta m'$,

$f_P(m_1, \dots, m_k) = \perp \Rightarrow \tau_{P'} m_1' \dots m_k'$ does not have a head normal form.

***D e f i n i t i o n 2 .*** We say, that an interpretation algorithm A can improve after the translation, if there exist such typed program $P$ of the form (1), $m_1, ..., m_k \in M$, and canonical notion of $\delta$-reduction, that A works infinitely on $P$ and on the term $F_1(m_1, \dots, m_k)$, and A stops on the untyped program $P'$ of the form (3) and on the term $F_1' m_1' \dots m_k'$ with a result $m'$, where $m \in M$, $m \neq \perp$.

***D e f i n i t i o n 3 .*** We say, that an interpretation algorithm A can deteriorate after the translation, if there exist such typed program $P$ of the form (1), $m_1, \dots, m_k \in M$, and canonical notion of $\delta$-reduction, that A stops on $P$ and on the term $F_1(m_1, \dots, m_k)$ with a result $m$, where $m \in M, m \neq \perp$, and $A$ works infinitely on the untyped program $P'$ of the form (3) and on the term $F_1' m_1' \dots m_k'$.

We recall the following notation for some untyped terms to be used in what follows: $I \equiv \lambda x.x, T \equiv \lambda xy.x$, $F \equiv \lambda xy.y$, $\Omega \equiv (\lambda x.xx)(\lambda x.xx)$, if $t_1$ then $t_2$ else $t_3 \equiv t_1 t_2 t_3$, Zero $\equiv \lambda x.xT, \perp' \equiv \Omega$, $0' \equiv I, (n+1)' \equiv \lambda x.xFn'$, where $x, y \in V$, $t_1, t_2, t_3 \in \Lambda$, $n \in N$.

It is easy to see that: the term $\Omega$ does not have a head normal form, if $T$ then $t_2$ else $t_3 \twoheadrightarrow_\beta t_2$, if $F$ then $t_2$ else $t_3 \twoheadrightarrow_\beta t_3$, Zero $0' \twoheadrightarrow_\beta T$, Zero $(n+1)' \twoheadrightarrow_\beta F$, Zero$\perp'$ does not have a head normal form, term $n'$ is closed normal form, and if $n_1 \neq n_2$, then $n_1'$ and $n_2'$ are not congruent terms, where $n, n_1, n_2 \in N$.

***T h e o r e m 15.*** Interpretation algorithm FSRNF can deteriorate after the translation, but cannot improve.

***P r o o f .*** Interpretation algorithm FSRNF cannot improve after the translation, since FSRNF is complete for typed programs, Theorem 4. Let us show, that FSRNF can deteriorate after the translation. Let $M = N \cup \{\perp\}$, $x \in V_M^T$, $F_1, F_2 \in V_{[M \to M]}^T$.

Program $P_6$ is: $F_1 = \lambda x[F_2(\perp)]$,
$F_2 = \lambda x[0]$.

Program $P_6{}'$ is: $F_1 = \lambda x.F_2 \Omega$,
$F_2 = \lambda x.0'$,

where $x, F_1, F_2 \in V$.

Let us show, that $(1, 0) \in Proc_{\text{FSRNF}}(P_6)$ and $(1', 0') \notin Proc_{\text{FSRNF}}(P_6')$.

For $P_6$ and FSRNF we have: $F_1(1); \lambda x[F_2(\perp)](1) \to_\beta F_2(\perp); \lambda x[0](\perp) \to_\beta 0$.

For $P_6'$ and FSRNF we have: $F_1 1'; (\lambda x.F_2 \Omega) 1' \to_\beta F_2 \Omega \to_\beta F_2 \Omega \to_\beta \dots$ and so on. □

***Theorem 16.*** Interpretation algorithm LSRNF can both improve and deteriorate after the translation.

***Proof.*** Let $M = N \cup \{\perp\}$, $x \in V_M^T$, $F_1, F_2, F_3 \in V_{[M \to M]}^T$.

Program $P_7$ is: $F_1 = \lambda x[sg2(F_2(x), F_3(x))]$,
$F_2 = \lambda x[F_2(x)]$,
$F_3 = \lambda x[0]$.

Program $P_7'$ is: $F_1 = \lambda x.Sg2(F_2 x)(F_3 x)$,
$F_2 = \lambda x.F_2 x$,
$F_3 = \lambda x.0'$,

where $Sg2 \equiv \lambda xy.$ if Zero $y$ then $0'$ else $1'$, $x, y, F_1, F_2, F_3 \in V$.

Let us show, that $(1,0) \notin Proc_{\text{LSRNF}}(P_7)$ and $(1',0') \in Proc_{\text{LSRNF}}(P_7')$.

For $P_7$ and LSRNF we have: $F_1(1); \lambda x[sg2(F_2(x), F_3(x))](1) \to_\beta sg2(F_2(1), F_3(1)); sg2(\lambda x[F_2(x)](1), F_3(1)) \to_\beta sg2(F_2(1), F_3(1)); \ldots$ and so on.

For $P_7'$ and LSRNF we have: $F_1 1'; (\lambda x.Sg2(F_2 x)(F_3 x))1' \to_\beta Sg2(F_2 1')(F_3 1') \equiv (\lambda xy.$ if Zero $y$ then $0'$ else $1')(F_2 1')(F_3 1') \to_\beta (\lambda y.$ if Zero $y$ then $0'$ else $1')(F_3 1') \to_\beta$ if Zero $(F_3 1')$ then $0'$ else $1' \equiv$ if $(\lambda x.xT)(F_3 1')$ then $0'$ else $1' \to_\beta$ if $(F_3 1')T$ then $0'$ else $1'$; if $((\lambda x.0')1')T$ then $0'$ else $1' \to_\beta$ if $0'T$ then $0'$ else $1' \to_\beta$ if $T$ then $0'$ else $1' \to\to_\beta 0'$.

It is easy to see that $(1,0) \in Proc_{\text{LSRNF}}(P_6)$ and $(1',0') \notin Proc_{\text{LSRNF}}(P_6')$. □

***Theorem 17.*** Interpretation algorithm PAS can improve after the translation, but cannot deteriorate.

***Proof.*** Interpretation algorithm PAS cannot deteriorate after the translation, since PAS is complete for untyped programs, Theorem 11. It is easy to see that $(1,0) \notin Proc_{\text{PAS}}(P_7)$ and $(1',0') \in Proc_{\text{PAS}}(P_7')$. Therefore, PAS can improve after the translation. □

***Theorem 18.*** Interpretation algorithm ACT can deteriorate after the translation, but cannot improve.

***Proof.*** First we prove that ACT can deteriorate after the translation. Let $M = N \cup \{\perp\}$, $x \in V_M^T$, $F_1, F_2 \in V_{[M \to M]}^T$.

Program $P_8$ is: $F_1 = \lambda x[sg1(F_2(x), F_3(x))]$,
$F_2 = \lambda x[0]$,
$F_3 = \lambda x[F_3(x)]$.

Program $P_8'$ is: $F_1 = \lambda x.Sg1(F_2 x)(F_3 x)$,
$F_2 = \lambda x.0'$,
$F_3 = \lambda x.F_3 x$,

where $Sg1 \equiv \lambda xy.$ if Zero $x$ then $0'$ else $1'$, $x, y, F_1, F_2, F_3 \in V$.

Let us show, that $(1,0) \in Proc_{\text{ACT}}(P_8)$ and $(1',0') \notin Proc_{\text{ACT}}(P_8')$.

For $P_8$ and ACT we have: $F_1(1); \lambda x[sg1(F_2(x), F_3(x))](1) \to_\beta$
$sg1(F_2(1), F_3(1)); sg1(\lambda x[0](1), F_3(1)) \to_\beta sg1(0, F_3(1)) \to_\delta 0$.

For $P_8'$ and ACT we have: $F_1 1'; (\lambda x.Sg1(F_2 x)(F_3 x))1' \to_\beta Sg1(F_2 1')(F_3 1') \equiv (\lambda xy.$if Zero $x$ then $0'$ else $1')(F_2 1')(F_3 1')$, now we must apply algorithm ACT to the term $F_2 1'; (\lambda x.0')1' \to_\beta 0'$; further, we have $(\lambda xy.$if Zero $x$ then $0'$ else $1')0'(F_3 1') \to_\beta$

$(\lambda y.\text{if Zero } 0' \text{ then } 0' \text{ else } 1')(F_3 1')$, now we must apply the algorithm ACT to the $F_3 1'; (\lambda x.F_3 x)1' \rightarrow_\beta F_3 1'; \ldots$ and so on.

Now we prove that ACT cannot improve after the translation. We assume to the contrary that for some typed program $P$ of the form (1), $m_1, \ldots, m_k \in M$, and for some canonical notion of $\delta$-reduction ACT works infinitely on $P$ and $F_1(m_1, \ldots, m_k)$, and ACT stops on the untyped program $P'$ of the form (3) and on the term $F_1' m_1' \ldots m_k'$ with a result $m'$, where $m \in M$, $m \neq \perp$. Therefore, $(m_1', \ldots, m_k', m') \in Proc_{\text{ACT}}(P')$ and according to Theorem 10 we have $(m_1', \ldots, m_k', m') \in Fix(P')$ and $\tau_{P'} m_1' \ldots m_k' \twoheadrightarrow_\beta m'$. According to Theorem 14 $f_P(m_1, \ldots, m_k) = m \neq \perp$. According to the translation algorithm and the property of algorithm ACT, that each "argument must be counted", when ACT is applied to $P'$ and $F_1' m_1' \ldots m_k'$, ACT will stop on $P$ and $F_1(m_1, \ldots, m_k)$ (with the result $m$, Theorem 3), that is a contradiction. $\square$

*Received 14.11.2017*

## REFERENCES

1. **Nigiyan S.A.** Functional Languages. // Programming and Computer Software, 1992, v. 17, № 5, p. 290–297.
2. **Nigiyan S.A.** On Interpretation of Functional Programming Languages. // Programming and Computer Software, 1993, v. 19, № 2, p. 71–78.
3. **Nigiyan S.A.** On Non-classical Theory of Computability. // Proceedings of the YSU. Physical and Mathematical Sciences, 2015, № 1, p. 52–60.
4. **Nigiyan S.A.** On $\lambda$-Definability of Arithmetical Functions with Indeterminate Values of Arguments. // Proceedings of the YSU. Physical and Mathematical Sciences, 2016, № 2, p. 39–47.
5. **Nigiyan S.A., Khondkaryan T.V.** On Translation of Typed Functional Programs into Untyped Functional Programs. // Proceedings of the YSU. Physical and Mathematical Sciences, 2017, v. 51, № 2, p. 177–186.
6. **Nigiyan S.A., Khondkaryan T.V.** On Canonical Notion of $\delta$-Reduction and on Translation of Typed $\lambda$-Terms into Untyped $\lambda$-Terms. // Proceedings of the YSU. Physical and Mathematical Sciences, 2017, v. 51, № 1, p. 46–52.
7. **Barendregt H.** The Lambda Calculus. Its Syntax and Semantics. North-Holland Publishing Company, 1981.
8. **Nigiyan S.A., Avetisyan S.A.** Semantics of Untyped Functional Programs. // Programming and Computer Software, 2002, v. 28, № 3, p. 119–126.
9. **Budaghyan L.E.** Formalizing the Notion of $\delta$-Reduction in Monotonic Models of Typed $\lambda$-Calculus. // Algebra, Geometry & Their Applications, 2002, v. 1, p. 48–57.
10. **Hakopian R.Yu.** On Procedural Semantics of Strong Typed Functional Programs. // Proceedings of YSU, 2008, № 3, p. 59–69 (in Russian).
11. **Manna Z.** Mathematical Theory of Computation. McGraw-Hill Book Company, 1974.
12. **Hrachyan G.G.** On Basic Semantics of Untyped Functional Programs. // Programming and Computer Software, 2009, v. 35, № 3, p. 121–135.