PROCEEDINGS OF THE YEREVAN STATE UNIVERSITY

Physical and Mathematical Sciences 2018, **52**(2), p. 109–118

Informatics

ON INCOMPARABILITY OF INTERPRETATION ALGORITHMS OF TYPED FUNCTIONAL PROGRAMS WITH RESPECT TO UNDEFINED VALUE

D. A. GRIGORYAN *

Chair of Programming and Information Technologies YSU, Armenia

In the paper the interpretation algorithms of typed functional programs are considered. The interpretation algorithm is based on substitution, β-reduction and canonical δ-reduction. It is shown that seven known interpretation algorithms (FS (of full substitution), PES (of parallel external substitution), LES (of left external substitution), PIS (of parallel inner substitution), LIS (of left inner substitution), ACT (active algorithm), PAS (passive algorithm)) are pairwise incomparable with respect to undefined value ($\perp$-incomparable).

MSC2010: 68N18.

***Keywords*:** typed functional program, canonical δ-reduction, interpretation algorithm, $\perp$-incomparability.

Typed λ-Terms, Canonical Notion of δ-Reduction, Typed Functional Programs. The definitions of this section can be found in [1–3]. Let M be a partially ordered set, which has a least element $\perp$, which corresponds to the indeterminate value, and each element of M is comparable only with $\perp$ and itself. Let us define the set of types (denoted by Types):

1. $M \in Types$;

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

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

Let $\alpha \in Types$ and V_α be a countable set of variables of type α, then $V = \bigcup\limits_{\alpha \in Types} V_\alpha$ is the set of all variables. The set of all terms, denoted by $\Lambda = \bigcup\limits_{\alpha \in Types} \Lambda_\alpha$, where Λ_α is the set of terms of type α, is defined as follows:

* E-mail: david.grigoryan.a@gmail.com

1. if $c \in \alpha, \alpha \in Types$, then $c \in \Lambda_\alpha$;

2. if $x \in V_\alpha$, $\alpha \in Types$, then $x \in \Lambda_\alpha$;

3. if $\tau \in \Lambda_{[\alpha_1 \times ... \times \alpha_k \to \beta]}$, $t_i \in \Lambda_{\alpha_i}$, where $\beta, \alpha_i \in Types$, $i = 1,...,k$, $k \geq 1$, then $\tau(t_1,...,t_k) \in \Lambda_\beta$ (the operation of application, $(t_1,...,t_k)$ is the scope of the applicator τ);

4. if $\tau \in \Lambda_\beta$, $x_i \in V_{\alpha_i}$ where $\beta, \alpha_i \in Types$, $i \neq j \implies x_i \neq x_j$, $i, j = 1,...,k$, $k \geq 1$ then $\lambda x_1 ... x_k[\tau] \in \Lambda_{[\alpha_1 \times ... \times \alpha_k \to \beta]}$ (the operation of abstraction, τ is the scope of the abstractor $\lambda x_1 ... x_k$).

The notion of free and bound occurrences of variables as well as free and bound variable are introduced in the conventional way. The set of all free variables in the term t is denoted by *FV(t)*. 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. The free occurrence of a variable in the term is called internal if it does not enter in the applicator, whose scope contains a free occurrence of some variable. The free occurrence of a variable in the term is called external if it does not enter in the scope of the applicator that contains a free occurrence of some variable.

Let $t \in \Lambda_\alpha, \alpha \in Types$ and $FV(t) \subset \{y_1,...,y_n\}, \bar{y}_0 = (y_1^0,...,y_n^0)$, where $y_i \in V_{\beta_i}$, $y_i^0 \in \beta_i$, $\beta_i \in Types$, $i = 1,...,n$, $n \geq 0$. The value of the term t for the values of the variables $y_1,...y_n$ equal to $\bar{y}_0 = (y_1^0,...,y_n^0)$, is denoted by $Val_{\bar{y}_0}(t)$ and is defined in the conventional way.

Let terms $t_1, t_2 \in \Lambda_\alpha$, $\alpha \in Types$, $FV(t_1) \cup FV(t_2) = \{y_1,...,y_n\}$, $y_i \in V_{\beta_i}$, $\beta_i \in Types$, $i = 1,...,n$, $n \geq 0$, then terms t_1 and t_2 are called equivalent (denoted by $t_1 \sim t_2$) if for any $\bar{y}_0 = (y_1^0,...,y_n^0)$, where $y_i^0 \in V_{\beta_i}$, $i = 1,...,n$ we have: $Val_{\bar{y}_0}(t_1) = Val_{\bar{y}_0}(t_2)$. A term $t \in \Lambda_\alpha, \alpha \in Types$, is called a constant term with value $a \in \alpha$ if $t \sim a$.

Further, we assume that M is a recursive set and considered terms use variables of any order and constants of order ≤ 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 at value $f(m_1,...,m_k) \in M$ for all $m_1,...,m_k \in M$ [2].

A term $t \in \Lambda$ with a fixed occurrence of a subterm $\tau_1 \in \Lambda_\alpha$, where $\alpha \in Types$, is denoted by t_{τ_1} and a term with this occurrence but τ_1 replaced by τ_2, where $\tau_2 \in \Lambda_\alpha$, is denoted by t_{τ_2}. To show mutually different variables of interest $x_1,...,x_k$, $k \geq 1$, of a term t, the notation $t[x_1,...,x_k]$ is used. The notation $t[t_1,...,t_k]$ denotes the term obtained by the simultaneous substitution of the terms $t_1,...,t_k$ for all free occurrences of the variables $x_1,...,x_k$ respectively, where $x_i \in V_{\alpha_i}, i \neq j \Rightarrow x_i \not\equiv x_j, t_i \in \Lambda_{\alpha_i}$, $\alpha_i \in Types$, $i, j = 1,..,k$, $k \geq 1$. 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 of the form $\lambda x_1 ... x_k[\tau[x_1,...,x_k]](t_1,...,t_k)$, where $x_i \in V_\alpha, i \neq j \Rightarrow x_i \not\equiv x_j, \tau \in \Lambda, t_i \in \Lambda_{\alpha_i}, \alpha_i \in Types$, $i, j = 1,...,k$, $k \geq 1$, is called a β-redex. Its convolution is the term $\tau[t_1,...,t_k]$. The set of all pairs (τ_0, τ_1), where τ_0 is a β-redex and τ_1 is

its convolution, is called a notion of β-reduction and is denoted by β. A one-step β-reduction ($\to_\beta$) and β-reduction ($\to\to_\beta$) are defined in the conventional way. A term containing no β-redexes is called a β-normal form. The set of all β-normal forms is denoted by β-NF.

δ-redex has a form $f(t_1,...,t_k)$, where $f \in [M^k \to M]$, $t_i \in \Lambda_M$, $i = 1,...,k$, $k \geq 1$, and its convolution is either $m \in M$ and in this case $f(t_1,...,t_k) \sim m$ or a subterm t_i and in this case $f(t_1,...,t_k) \sim t_i$, $i = 1,...,k$. A fixed set of term pairs (τ_0, τ_1), where τ_0 is a δ-redex and τ_1 is its convolution, is called a notion of δ-reduction and is denoted by δ. A one-step δ-reduction ($\to_\delta$) and δ-reduction ($\to\to_\delta$) are defined in the conventional way.

A one-step $\beta\delta$-reduction ($\to_{\beta\delta}$) and $\beta\delta$-reduction ($\to\to_{\beta\delta}$) defined in the conventional way. A term containing no $\beta\delta$-redexes is called normal form. The set of all normal forms is denoted by NF.

A notion of δ-reduction is called a single-valued notion of δ-reduction, if δ is a single-valued relation, i.e. if $(\tau_0, \tau_1) \in \delta$ and $(\tau_0, \tau_2) \in \delta$, then $\tau_1 \equiv \tau_2$, where $\tau_0, \tau_1, \tau_2 \in \Lambda_M$. A notion of δ-reduction is called an effective notion of δ-reduction if there exists an algorithm, which for any term $f(t_1,...,t_k)$, where $f \in [M^k \to M]$, $t_i \in \Lambda_M, i = 1,...,k$, $k \geq 1$, gives its convolution if $f(t_1,...,t_k)$ is a δ-redex and stops with a negative answer otherwise.

Definition 1. [3] An effective, single-valued notion of δ-reduction is called a canonical notion of δ-reduction if:

1. $t \in \beta$-NF, $t \sim m$, $m \in M \setminus \{\bot\} \Rightarrow t \to\to_\delta m$;
2. $t \in \beta$-NF, $FV(t) = \varnothing$, $t \sim \bot \Rightarrow t \to\to_\delta \bot$;

Typed functional program P is the following system of equations:

$$P \begin{cases} F_1 = t_1[F_1,...,F_n], \\ \quad ... \\ F_n = t_n[F_1,...,F_n], \end{cases} \tag{1}$$

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

Every typed functional program P has the least solution. Let $(f_1,...,f_n) \in \alpha_1 \times ... \times \alpha_n$ 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 [1].

$Fix(P) = \{(m_1,...,m_k,m) |\, f_p(m_1,...,m_k) = m$, where $m, m_1,...,m_k \in M, k \geq 1\}$.

Interpretation Algorithms, $\bot$-Incomparability. The input of the interpretation algorithm A is a program P of the form (1), a term $F_1(m_1,...,m_k)$, where $m_i \in M$, $i = 1,...,k$, $k \geq 1$, and a canonical notion of δ-reduction. The algorithm A stops with result $m \in M$ or works infinitely. The algorithm A uses three kinds of operations: substitution of the terms $t_1,...,t_n$ instead of some free occurrences of variables $F_1,...,F_n$, one-step β-reduction and one-step δ-reduction.

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

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

Theorem 1. Every interpretation algorithm A is consistent.

Proof. Follows from the results of [4]. □

Definition 2. Let A and B be interpretation algorithms, then $A \prec_\perp B$, if for any program P, any canonical notion of δ-reduction and any $m_1,...,m_k \in M$, $k \geq 1$ we have: if $(m_1,...,m_k,\perp) \in Proc_A(P)$, then $(m_1,...,m_k,\perp) \in Proc_B(P)$.

Definition 3. Interpretation algorithms A and B are $\perp$-incomparable, if $A \not\prec_\perp B$ and $B \not\prec_\perp A$.

To show a sequence $F_{i_1},...,F_{i_s}$, $s \geq 1$, of some free occurences of variables of the set $\{F_1,...,F_n\}$ in the term t, the notion $t < F_{i_1},...,F_{i_s} >$ is used. The notion $t < t_{i_1},...,t_{i_s} >$ denotes the term obtained by the simultaneous substitution of the terms $t_{i_1},...,t_{i_s}$ for free occurences $F_{i_1},...,F_{i_s}$ respectively.

Definition 4. (Interpretation algorithms) Input of the interpretation algorithms FS, PES, LES, PIS, LIS, PAS, ACT is a program P of the form (1), term $t \in \Lambda$ and canoncial notion of δ-reduction.

STEP 1:

FS, PES, LES, PIS, LIS, PAS, ACT: If $t \in NF$ and $FV(t) \cap \{F_1,...,F_n\} = \varnothing$ then t. else go to STEP 2.

STEP 2:

FS, PES, LES, PIS, LIS: If $t \equiv t_\tau$ where τ is leftmost redex (δ-redex or β-redex), then $A(P,t_{\tau'})$, where τ' is the convolution of the τ, $A \in \{$FS, PES, LES, PIS, LIS$\}$, else go to STEP 3.

ACT, PAS: If $t \equiv t_{F_i}$, $0 \leq i \leq n$, where t_{F_i} is the term t with a fixed leftmost free occurrence of a variable of the set $\{F_1,...,F_n\}$ which is located to the left of the leftmost redex, then $A(P,t_{t_i})$, where $A \in \{\text{ACT}, \text{PAS}\}$, else go to STEP 3.

STEP 3:

FS: If $t \equiv t[F_1,...,F_n]$, then $FS(P,t[t_1,...,t_n])$.

PES: If $t \equiv t < F_{i_1},...,F_{i_k} >$, where $F_{i_1},...,F_{i_k}, k > 0$ is the sequence of all external free occurrences of variables of the set $\{F_1,...,F_n\}$, then $\text{PES}(P,t < t_{i_1},...,t_{i_k} >)$.

LES: If $t \equiv t_{F_i}$, where F_i is the leftmost external free occurrence of a variable of the set $\{F_1,...,F_n\}$, then $\text{LES}(P,t_{t_i})$.

PIS: If $t \equiv t < F_{i_1},...,F_{i_k} >$, where $F_{i_1},...,F_{i_k}, k > 0$ is the sequence of all internal free occurrences of variables of the set $\{F_1,...,F_n\}$, then $\text{PIS}(P,t < t_{i_1},...,t_{i_k} >)$.

LIS: If $t \equiv t_{F_i}$, where F_i is the leftmost internal free occurrence of a variable of the set $\{F_1,...,F_n\}$, then $\text{LIS}(P,t_{t_i})$.

ACT: If $t \equiv t_\tau$, where $\tau \equiv \lambda x_1...x_r[\tau[x_1,...,x_r]](\tau_1,...,\tau_r)$ and τ is leftmost redex, then $\text{ACT}(P,t_{\tau[\text{ACT}(P,\tau_1),...,\text{ACT}(P,\tau_r)]})$, else go to step 4.

PAS: If $t \equiv t_\tau$, where $\tau \equiv \lambda x_1...x_r[\tau[x_1,...,x_r]](\tau_1,...,\tau_r)$ and τ is leftmost redex, then $\text{PAS}(P,t_{\tau[\tau_1,...,\tau_r]})$, else go to STEP 4.

STEP 4:

ACT, PAS: If $t \equiv t_\tau$, where τ is leftmost redex, which is a δ-redex, then $A(P,t_{\tau'})$, where τ' is the convolution of the τ, $A \in \{$ACT, PAS$\}$.

Theorem 2. Interpretation algorithms FS, PES, LES, PIS, LIS, PAS and ACT are pairwise $\perp$-incomparable.

Proof. Let us fix $M = N \cup \{\perp\}$, where $N = \{0,1,2,...\}$ and $C = \{not\_eq, numbers\}$, where $not\_eq, numbers \in [M^2 \to M]$ are built-in functions and for every $m_1, m_2 \in M$, we have:

$$not\_eq(m_1,m_2) = \begin{cases} 1, & m_1,m_2 \in N,\ m_1 \neq m_2, \\ \perp, & \text{otherwise}, \end{cases}$$

$$numbers(m_1,m_2) = \begin{cases} 1, & m_1,m_2 \in N, \\ \perp, & \text{otherwise}. \end{cases}$$

It is easy to see that *not_eq* and *numbers* are strongly computable, naturally extended functions with indeterminate values of arguments. Let us fix canonical notions of δ-reduction δ for the set C.

δ is: $(not\_eq(n_1,n_2),1) \in \delta$, where $n_1,n_2 \in N$ and $n_1 \neq n_2$
$(not\_eq(t,t),\perp) \in \delta$, where $t \in \Lambda_M$
$(not\_eq(t,\perp),\perp) \in \delta$, where $t \in \Lambda_M$
$(not\_eq(\perp,t),\perp) \in \delta$, where $t \in \Lambda_M$
$(numbers(n_1,n_2),1) \in \delta$, where $n_1,n_2 \in N$
$(numbers(\perp,t),\perp) \in \delta$, where $t \in \Lambda_M$
$(numbers(t,\perp),\perp) \in \delta$, where $t \in \Lambda_M$
$(numbers(numbers(t_1,t_2),numbers(t_2,t_1)),numbers(t_2,t_1)) \in \delta$,
where $t_1,t_2 \in \Lambda_M$.

Let $x,y,z \in V_M, F,F_1,F_2,F_3,F_4,F_5 \in V_{[M\to M]}$.

To show that $A \not\prec_\perp B$, where $A \in \{$LES, PAS, LIS, ACT$\}$, $B \in \{$FS, PES, PIS$\}$ we take program P_1:

$$P_1 \begin{cases} F_1 = \lambda x[not\_eq(F_2(x),F_3(x))] \\ F_2 = \lambda x[F_3(x)] \\ F_3 = \lambda x[F_2(x)] \end{cases}$$

For LES, PAS, ACT, LIS we have: $F_1(0)$; $\lambda x[not\_eq(F_2(x),F_3(x))](0) \to_\beta$
$not\_eq(F_2(0),F_3(0))$;$not\_eq(\lambda x[F_3(x)](0),F_3(0)) \to_\beta not\_eq(F_3(0),F_3(0)) \to_\delta \perp$;
For FS, PES, PIS we have:
$F_1(0)$;$\lambda x[not\_eq(F_2(x),F_3(x))](0) \to_\beta not\_eq(F_2(0),F_3(0))$;
$not\_eq(\lambda x[F_3(x)](0),\lambda x[F_2(x)](0)) \to\to_\beta not\_eq(F_3(0),F_2(0))$;
$not\_eq(\lambda x[F_2(x)](0),\lambda x[F_3(x)](0)) \to\to_\beta not\_eq(F_2(0),F_3(0))$; and so on.

To show that PIS $\not\prec_\perp$ LIS, PIS $\not\prec_\perp$ ACT we take program P_2:

$$P_2 \begin{cases} F_1 = \lambda x[not\_eq(F_2(x),F_3(x))] \\ F_2 = \lambda x[F_2(x)] \\ F_3 = \lambda x[F_2(x)] \end{cases}$$

For PIS we have: $F_1(0)$;$\lambda x[not\_eq(F_2(x),F_3(x))](0) \to_\beta not\_eq(F_2(0),F_3(0))$;
$not\_eq(\lambda x[F_2(x)](0),\lambda x[F_2(x)](0)) \to\to_\beta not\_eq(F_2(0),F_2(0)) \to_\delta \perp$.

For LIS, ACT we have: $F_1(0); \lambda x[not\_eq(F_2(x), F_3(x))](0) \to_\beta not\_eq(F_2(0), F_3(0))$; $not\_eq(\lambda x[F_2(x)](0), F_3(0)) \to_\beta not\_eq(F_2(0), F_3(0))$; and so on.

To show that LIS $\not\prec_\perp$ ACT we take program P_3:

$$P_3 \begin{cases} F_1 = \lambda x[not\_eq(F_3(x), F_2(x))] \\ F_2 = \lambda y[\lambda x[x](F_2(y))] \\ F_3 = \lambda y[\lambda x[x](F_2(y))] \end{cases}$$

For LIS we have: $F_1(0); \lambda x[not\_eq(F_3(x), F_2(x))](0) \to_\beta not\_eq(F_3(0), F_2(0))$;
$not\_eq(\lambda y[\lambda x[x](F_2(y))](0), F_2(0)) \to\to_\beta not\_eq(F_2(0), F_2(0)) \to_\delta \perp$.
For ACT we have: $F_1(0); \lambda x[not\_eq(F_3(x), F_2(x))](0) \to_\beta not\_eq(F_3(0), F_2(0))$;
$not\_eq(\lambda y[\lambda x[x](F_2(y))](0), F_2(0)) \to_\beta not\_eq(\lambda x[x](F_2(0)), F_2(0))$;
Now we must apply ACT to the term $F_2(0); \lambda y[\lambda x[x](F_2(y))](0) \to_\beta \lambda x[x](F_2(0))$;
Now we must apply ACT to the term $F_2(0)$ and so on.

To show that ACT $\not\prec_\perp$ LIS we take program P_4:

$$P_4 \begin{cases} F_1 = \lambda z[not\_eq(F_2(z), \lambda x[0](F_2(z)))] \\ F_2 = \lambda y[\lambda F[F(F_2(y))](F_3)] \\ F_3 = \lambda x[0] \end{cases}$$

For ACT we have: $F_1(0); \lambda z[not\_eq(F_2(z), \lambda x[0](F_2(z)))](0) \to_\beta$
$not\_eq(F_2(0), \lambda x[0](F_2(0)))$; $not\_eq(\lambda y[\lambda F[F(F_2(y))](F_3)](0), \lambda x[0](F_2(0))) \to_\beta$
$not\_eq(\lambda F[F(F_2(0))](F_3), \lambda x[0](F_2(0)))$;
$not\_eq(\lambda F[F(F_2(0))](\lambda x[0]), \lambda x[0](F_2(0))) \to_\beta$
$not\_eq(\lambda x[0](F_2(0)), \lambda x[0](F_2(0))) \to_\delta \perp$.
For LIS we have: $F_1(0); \lambda z[not\_eq(F_2(z), \lambda x[0](F_2(z)))](0) \to\to_\beta not\_eq(F_2(0), 0)$;
$not\_eq(\lambda y[\lambda F[F(F_2(y))](F_3)](0), 0) \to\to_\beta not\_eq(F_3(F_2(0)), 0)$;
$not\_eq(F_3(\lambda y[\lambda F[F(F_2(y))](F_3)](0)), 0) \to\to_\beta not\_eq(F_3(F_3(F_2(0))), 0)$;
$not\_eq(F_3(F_3(\lambda y[\lambda F[F(F_2(y))](F_3)](0))), 0) \to\to_\beta$
$not\_eq(F_3(F_3(F_3(F_2(0)))), 0)$; and so on.

Let $A \in \{\text{PIS, LIS}\}$ and $B \in \{\text{PES, LES, PAS}\}$. To show that $A \not\prec_\perp B$, we take program P_5:

$$P_5 \begin{cases} F_1 = \lambda x[not\_eq(F_3(F_2(x)), F_3(F_3(x)))] \\ F_2 = \lambda x[F_3(x)] \\ F_3 = \lambda x[F_3(x)] \end{cases}$$

For PIS we have: $F_1(0); \lambda x[not\_eq(F_3(F_2(x)), F_3(F_3(x)))](0) \to_\beta$
$not\_eq(F_3(F_2(0)), F_3(F_3(0)))$; $not\_eq(F_3(\lambda x[F_3(x)](0)), F_3(\lambda x[F_3(x)](0))) \to\to_\beta$
$not\_eq(F_3(F_3(0)), F_3(F_3(0))) \to_\delta \perp$;
For LIS we have: $F_1(0); \lambda x[not\_eq(F_3(F_2(x)), F_3(F_3(x)))](0) \to_\beta$
$not\_eq(F_3(F_2(0)), F_3(F_3(0)))$; $not\_eq(F_3(\lambda x[F_3(x)](0)), F_3(F_3(0))) \to_\beta$
$not\_eq(F_3(F_3(0)), F_3(F_3(0))) \to_\delta \perp$;
For PES we have: $F_1(0); \lambda x[not\_eq(F_3(F_2(x)), F_3(F_3(x)))](0) \to_\beta$
$not\_eq(F_3(F_2(0)), F_3(F_3(0)))$; $not\_eq(\lambda x[F_3(x)](F_2(0)), \lambda x[F_3(x)](F_3(0))) \to\to_\beta$
$not\_eq(F_3(F_2(0)), F_3(F_3(0)))$; and so on.

For LES, PAS we have: $F_1(0);\lambda x[not\_eq(F_3(F_2(x)),F_3(F_3(x)))](0) \to_\beta$
$not\_eq(F_3(F_2(0)),F_3(F_3(0)));not\_eq(\lambda x[F_3(x)](F_2(0)),F_3(F_3(0))) \to_\beta$
$not\_eq(F_3(F_2(0)),F_3(F_3(0)))$; and so on.

To show that $A \not\leq_\perp B$, where $A \in \{\text{PES, LES, PAS}\}$, $B \in \{\text{PIS, LIS}\}$, we take program P_6:

$$P_6 \begin{cases} F_1 = \lambda x[not\_eq(F_2(F_3(x)),F_3(F_3(x)))] \\ F_2 = \lambda x[F_3(x)] \\ F_3 = \lambda x[F_3(x)] \end{cases}$$

For PES we have: $F_1(0)$; $\lambda x[not\_eq(F_2(F_3(x)),F_3(F_3(x)))](0) \to_\beta$
$not\_eq(F_2(F_3(0)),F_3(F_3(0)));not\_eq(\lambda x[F_3(x)](F_3(0)),\lambda x[F_3(x)](F_3(0))) \to\to_\beta$
$not\_eq(F_3(F_3(0)),F_3(F_3(0))) \to_\delta \perp$
For LES, PAS we have: $F_1(0);\lambda x[not\_eq(F_2(F_3(x)),F_3(F_3(x)))](0) \to_\beta$
$not\_eq(F_2(F_3(0)),F_3(F_3(0)));not\_eq(\lambda x[F_3(x)](F_3(0)),F_3(F_3(0))) \to_\beta$
$not\_eq(F_3(F_3(0)),F_3(F_3(0))) \to_\delta \perp$;
For PIS we have: $F_1(0)$; $\lambda x[not\_eq(F_2(F_3(x)),F_3(F_3(x)))](0) \to_\beta$
$not\_eq(F_2(F_3(0)),F_3(F_3(0)))$; $not\_eq(F_2(\lambda x[F_3(x)](0)),F_3(\lambda x[F_3(x)](0))) \to\to_\beta$
$not\_eq(F_2(F_3(0)),F_3(F_3(0)))$; and so on.
For LIS we have: $F_1(0)$; $\lambda x[not\_eq(F_2(F_3(x)),F_3(F_3(x)))](0) \to_\beta$
$not\_eq(F_2(F_3(0)),F_3(F_3(0)));not\_eq(F_2(\lambda x[F_3(x)](0)),F_3(F_3(0))) \to_\beta$
$not\_eq(F_2(F_3(0)),F_3(F_3(0)))$; and so on.

To show that FS$\not\leq_\perp B$, where $B \in \{\text{PES, LES, PIS, LIS, PAS, ACT}\}$, we take program P_7:

$$P_7 \begin{cases} F_1 = \lambda x[not\_eq(F_2(F_3(x)),F_4(F_5(x)))] \\ F_2 = \lambda x[F_4(x)] \\ F_3 = \lambda x[F_5(x)] \\ F_4 = \lambda x[F_4(x)] \\ F_5 = \lambda x[F_5(x)] \end{cases}$$

For FS we have:
$F_1(0)$; $\lambda x[not\_eq(F_2(F_3(x)),F_4(F_5(x)))](0) \to_\beta not\_eq(F_2(F_3(0)),F_4(F_5(0)))$;
$not\_eq(\lambda x[F_4(x)](\lambda x[F_5(x)](0)),\lambda x[F_4(x)](\lambda x[F_5(x)](0))) \to\to_\beta$
$not\_eq(F_4(F_5(0)),F_4(F_5(0)))) \to_\delta \perp$.
For PES we have:
$F_1(0);\lambda x[not\_eq(F_2(F_3(x)),F_4(F_5(x)))](0) \to_\beta not\_eq(F_2(F_3(0)),F_4(F_5(0)))$;
$not\_eq(\lambda x[F_4(x)](F_3(0)),\lambda x[F_4(x)](F_5(0))) \to\to_\beta not\_eq(F_4(F_3(0)),F_4(F_5(0)))$;
$not\_eq(\lambda x[F_4(x)](F_3(0)),\lambda x[F_4(x)](F_5(0)))$; and so on.
For LES, PAS we have:
$F_1(0)$; $\lambda x[not\_eq(F_2(F_3(x)),F_4(F_5(x)))](0) \to_\beta$
$not\_eq(F_2(F_3(0)),F_4(F_5(0)));not\_eq(\lambda x[F_4(x)](F_3(0)),F_4(F_5(0))) \to_\beta$
$not\_eq(F_4(F_3(0)),F_4(F_5(0)));not\_eq(\lambda x[F_4(x)](F_3(0)),F_4(F_5(0)))$; and so on.
For LIS we have:
$F_1(0)$; $\lambda x[not\_eq(F_2(F_3(x)),F_4(F_5(x)))](0) \to_\beta not\_eq(F_2(F_3(0)),F_4(F_5(0)))$;

$not\_eq(F_2(\lambda x[F_5(x)](0)),F_4(F_5(0))) \to_\beta not\_eq(F_2(F_5(0)),F_4(F_5(0)))$;
$not\_eq(F_2(\lambda x[F_5(x)](0)),F_4(F_5(0)))$; and so on.
For PIS we have:
$F_1(0)$; $\lambda x[not\_eq(F_2(F_3(x)),F_4(F_5(x)))](0) \to_\beta not\_eq(F_2(F_3(0)),F_4(F_5(0)))$;
$not\_eq(F_2(\lambda x[F_5(x)](0)),F_4(\lambda x[F_5(x)](0))) \to\to_\beta not\_eq(F_2(F_5(0)),F_4(F_5(0)))$;
$not\_eq(F_2(\lambda x[F_5(x)](0)),F_4(\lambda x[F_5(x)](0)))$; and so on.
For ACT we have:
$F_1(0)$; $\lambda x[not\_eq(F_2(F_3(x)),F_4(F_5(x)))](0) \to_\beta not\_eq(F_2(F_3(0)),F_4(F_5(0)))$;
$not\_eq(\lambda x[F_4(x)](F_3(0)),F_4(F_5(0)))$; now me must apply algorithm ACT to the term
$F_3(0)$; $\lambda x[F_5(x)](0) \to_\beta F_5(0)$; $\lambda x[F_5(x)](0) \to_\beta F_5(0)$; and so on.

Let A =PES and $B \in$ {PS, LES, PAS, ACT}. To show that $A \not\to_\perp B$, we take program P_8:

$$P_8 \begin{cases} F_1 = \lambda x[not\_eq(F_2(F_3(x)),F_4(x))] \\ F_2 = \lambda x[F_2(x)] \\ F_3 = \lambda x[F_5(x)] \\ F_4 = \lambda x[F_2(F_3(x))] \\ F_5 = \lambda x[F_3(x)] \end{cases}$$

For PES we have: $F_1(0)$; $\lambda x[not\_eq(F_2(F_3(x)),F_4(x))](0) \to_\beta$
$not\_eq(F_2(F_3(0)),F_4(0))$; $not\_eq(\lambda x[F_2(x)](F_3(0)),\lambda x[F_2(F_3(x))](0)) \to\to_\beta$
$not\_eq(F_2(F_3(0)),F_2(F_3(0))) \to_\delta \perp$
For FS we have: $F_1(0)$; $\lambda x[not\_eq(F_2(F_3(x)),F_4(x))](0) \to_\beta$
$not\_eq(F_2(F_3(0)),F_4(0))$;
$not\_eq(\lambda x[F_2(x)](\lambda x[F_5(x)](0)),\lambda x[F_2(F_3(x))](0)) \to\to_\beta$
$not\_eq(F_2(F_5(0)),F_2(F_3(0)))$;
$not\_eq(\lambda x[F_2(x)](\lambda x[F_3(x)](0)),\lambda x[F_2(x)](\lambda x[F_5(x)](0))) \to\to_\beta$
$not\_eq(F_2(F_3(0)),F_2(F_5(0)))$;
$not\_eq(\lambda x[F_2(x)](\lambda x[F_5(x)](0)),\lambda x[F_2(x)](\lambda x[F_3(x)](0))) \to\to_\beta$
$not\_eq(F_2(F_5(0)),F_2(F_3(0)))$; and so on.
For LES, PAS we have:
$F_1(0)$; $\lambda x[not\_eq(F_2(F_3(x)),F_4(x))](0) \to_\beta not\_eq(F_2(F_3(0)),F_4(0))$;
$not\_eq(\lambda x[F_2(x)](F_3(0)),F_4(0)) \to_\beta not\_eq(F_2(F_3(0)),F_4(0))$; and so on.
For ACT we have:
$F_1(0)$; $\lambda x[not\_eq(F_2(F_3(x)),F_4(x))](0) \to_\beta not\_eq(F_2(F_3(0)),F_4(0))$;
$not\_eq(\lambda x[F_2(x)](F_3(0)),F_4(0))$; now we must apply ACT to the term $F_3(0)$;
$\lambda x[F_5(x)](0) \to_\beta F_5(0)$; $\lambda x[F_3(x)](0) \to_\beta F_3(0)$; now we must apply ACT to the term
$F_3(0)$; and so on.

Let A = ACT and $B \in$ {PES, LES, PAS}. To show that $A \not\to_\perp B$, we take program P_9:

$$P_9 \begin{cases} F_1 = \lambda x[not\_eq(F_2(F_3(x)),F_2(x))] \\ F_2 = \lambda x[F_2(x)] \\ F_3 = \lambda x[x] \end{cases}$$

For ACT we have: $F_1(0);\lambda x[not\_eq(F_2(F_3(x)),F_2(x))](0) \to_\beta$
$not\_eq(F_2(F_3(0)),F_2(0));not\_eq(\lambda x[F_2(x)](F_3(0)),F_2(0))$; now we must apply algorithm ACT to the term $F_3(0);\lambda x[x](0) \to_\beta 0$; Algorithm continues work on the term $not\_eq(F_2(0),F_2(0)) \to_\delta \perp$
For PES we have:
$F_1(0);\lambda x[not\_eq(F_2(F_3(x)),F_2(x))](0) \to_\beta not\_eq(F_2(F_3(0)),F_2(0))$;
$not\_eq(\lambda x[F_2(x)](F_3(0)),\lambda x[F_2(x)](0)) \to\to_\beta not\_eq(F_2(F_3(0)),F_2(0))$; and so on.
For LES, PAS we have:
$F_1(0);\lambda x[not\_eq(F_2(F_3(x)),F_2(x))](0) \to_\beta not\_eq(F_2(F_3(0)),F_2(0))$;
$not\_eq(\lambda x[F_2(x)](F_3(0)),F_2(0)) \to_\beta not\_eq(F_2(F_3(0)),F_2(0))$; and so on.

To show that PIS$\not\prec_\perp$ FS, we take program P_{10}:

$$P_{10}\begin{cases} F_1 = \lambda x[not\_eq(F_2(F_3(x)),F_4(x))] \\ F_2 = \lambda x[F_5(x)] \\ F_3 = \lambda x[F_5(x)] \\ F_4 = \lambda x[F_2(F_5(x))] \\ F_5 = \lambda x[F_2(x)] \end{cases}$$

For PIS we have: $F_1(0);\lambda x[not\_eq(F_2(F_3(x)),F_4(x))](0) \to_\beta$
$not\_eq(F_2(F_3(0)),F_4(0));not\_eq(F_2(\lambda x[F_5(x)](0)),\lambda x[F_2(F_5(x))](0)) \to\to_\beta$
$not\_eq(F_2(F_5(0)),F_2(F_5(0))) \to_\delta \perp$;
For FS we have: $F_1(0);\lambda x[not\_eq(F_2(F_3(x)),F_4(x))](0) \to_\beta$
$not\_eq(F_2(F_3(0)),F_4(0))$;
$not\_eq(\lambda x[F_5(x)](\lambda x[F_5(x)](0)),\lambda x[F_2(F_5(x))](0)) \to\to_\beta$
$not\_eq(F_5(F_5(0)),F_2(F_5(0)))$;
$not\_eq(\lambda x[F_2(x)](\lambda x[F_2(x)](0)),\lambda x[F_5(x)](\lambda x[F_2(x)](0))) \to\to_\beta$
$not\_eq(F_2(F_2(0)),F_5(F_2(0)))$;
$not\_eq(\lambda x[F_5(x)](\lambda x[F_5(x)](0)),\lambda x[F_2(x)](\lambda x[F_5(x)](0))) \to\to_\beta$
$not\_eq(F_5(F_5(0)),F_2(F_5(0)))$; and so on.

To show that LES $\not\prec_\perp$ PAS, we take program P_{11}:

$$P_{11}\begin{cases} F_1 = \lambda x[not\_eq(F_2(x),\lambda y[y](F_3(x)))] \\ F_2 = \lambda x[F_3(x)] \\ F_3 = \lambda x[F_3(x)] \end{cases}$$

For LES we have: $F_1(0);\lambda x[not\_eq(F_2(x),\lambda y[y](F_3(x)))](0) \to\to_\beta$
$not\_eq(F_2(0),F_3(0));not\_eq(\lambda x[F_3(x)](0),F_3(0)) \to_\beta not\_eq(F_3(0),F_3(0)) \to_\delta \perp$.
For PAS we have:
$F_1(0);\lambda x[not\_eq(F_2(x),\lambda y[y](F_3(x)))](0) \to_\beta not\_eq(F_2(0),\lambda y[y](F_3(0)))$;
$not\_eq(\lambda x[F_3(x)](0),\lambda y[y](F_3(0))) \to_\beta not\_eq(F_3(0),\lambda y[y](F_3(0)))$;
$not\_eq(\lambda x[F_3(x)](0),\lambda y[y](F_3(0))) \to_\beta not\_eq(F_3(0),\lambda y[y](F_3(0)))$; and so on.

To show that LES $\not\prec_\perp$ ACT and PAS$\not\prec_\perp$ACT, we take program P_{12}:

$$P_{12}\begin{cases} F_1 = \lambda x[not\_eq(F_2(F_3(x)),F_3(F_3(x)))] \\ F_2 = \lambda y[\lambda x[x](F_3(y))] \\ F_3 = \lambda y[\lambda x[x](F_2(y))] \end{cases}$$

For LES, PAS we have: $F_1(0);\lambda x[not\_eq(F_2(F_3(x)),F_3(F_3(x)))](0) \to_\beta$
$not\_eq(F_2(F_3(0)),F_3(F_3(0)));not\_eq(\lambda y[\lambda x[x](F_3(y))](F_3(0)),F_3(F_3(0))) \to\to_\beta$
$not\_eq(F_3(F_3(0)),F_3(F_3(0))) \to_\delta \bot$.
For ACT we have:
$F_1(0)$; $\lambda x[not\_eq(F_2(F_3(x)),F_3(F_3(x)))](0) \to_\beta not\_eq(F_2(F_3(0)),F_3(F_3(0)))$;
$not\_eq(\lambda y[\lambda x[x](F_3(y))](F_3(0)),F_3(F_3(0)))$; we must apply algorithm ACT to the term $F_3(0)$;
$\lambda y[\lambda x[x](F_2(y))](0) \to_\beta \lambda x[x](F_2(0))$; now we must algorithm apply ACT to the term $F_2(0)$;
$\lambda y[\lambda x[x](F_3(y))](0) \to_\beta \lambda x[x](F_3(0))$; and so on.

To show that PAS $\not\to_\bot$ LES, we take program P_{13}:

$$P_{13}\begin{cases} F_1 = \lambda x[numbers(numbers(F_2(x),F_3(x)),numbers(F_3(x),\lambda x[F_2(x)](0)))] \\ F_2 = \lambda x[not\_eq(x,0)] \\ F_3 = \lambda x[F_3(x)] \end{cases}$$

For PAS we have: $F_1(0)$;
$\lambda x[numbers(numbers(F_2(x),F_3(x)),numbers(F_3(x),\lambda x[F_2(x)](0)))](0) \to_\beta$
$numbers(numbers(F_2(0),F_3(0)),numbers(F_3(0),\lambda x[F_2(x)](0)))$;
$numbers(numbers(\lambda x[not\_eq(x,0)](0),F_3(0)),numbers(F_3(0),\lambda x[F_2(x)](0))) \to_\beta$
$numbers(numbers(not\_eq(0,0),F_3(0)),numbers(F_3(0),\lambda x[F_2(x)](0))) \to_\delta$
$numbers(\bot,numbers(F_3(0),\lambda x[F_2(x)](0))) \to_\delta \bot$.
For LES we have:
$F_1(0)$; $\lambda x[numbers(numbers(F_2(x),F_3(x)),numbers(F_3(x),\lambda x[F_2(x)](0)))](0) \to\to_\beta$
$numbers(numbers(F_2(0),F_3(0)),numbers(F_3(0),F_2(0))) \to_\delta numbers(F_3(0),F_2(0))$;
$numbers(\lambda x[F_3(x)](0),F_2(0)) \to_\beta numbers(F_3(0),F_2(0))$; and so on.

It is shown that for each pair of different algorithms $A,B \in \{$FS, PES, LES, PIS, LIS, PAS, ACT$\}$ the following holds: $A \not\to_\bot B$ and $B \not\to_\bot A$. Therefore, interpretation algorithms FS, PES, LES, PIS, LIS, PAS, ACT are pairwise $\bot$-incomparable. □

Received 07.05.2018

REFERENCES

1. **Nigiyan S.A.** Functional Languages. // Programming and Computer Software, 1992, v. 17, № 5, p. 290–297.
2. **Nigiyan S.A.** On Non-classical Theory of Computability. // Proceedings of the YSU. Physical and Mathematical Sciences, 2015, № 1, p. 52–60.
3. **Nigiyan S.A., Khondkaryan T.V.** On Canonical Notion of δ-Reduction and on Translation of Typed λ-Terms into Untyped λ-Terms. // Proceedings of the YSU. Physical and Mathematical Sciences, 2017, № 1, p. 46–52
4. **Hakopian R.Yu.** On Procedural Semantics of Strong Typed Functional Programs. // Proceedings of YSU, 2008, № 3, p. 59–69 (in Russian).