Mathematical Problems of Computer Science 49, 103–109, 2018.

On Dependence of Interpretation Algorithms of Typed Functional Programs on Canonical Notion of δ-Reduction

Davit A. Grigoryan

Department of Informatics and Applied Mathematics, YSU
e-mail: david.grigoryan.a@gmail.com

Abstract

In this paper the interpretation algorithms of typed functional programs are considered. The interpretation algorithm is based on substitutions, β-reduction and canonical δ-reduction. The basic semantics of typed functional program is a function with indeterminate values of arguments, which is the main component of its least solution. If the value of the basic semantics for some values of arguments is indeterminate, then the interpretation algorithm either stops with the value $\perp$, or works endlessly. It is shown that seven known interpretation algorithms are $\perp$-depend on canonical notion of δ-reduction. Here are these 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).

Keywords: Typed functional program, Canonical δ-reduction, Interpretation algorithm, $\perp$-dependency.

1. Typed λ-Terms, Canonical Notion of δ-Reduction, Typed Functional Programs

The definitions of this section can be found in [1, 2, 3]. Let M be a partially ordered set, which has a least 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 β,α_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 $ord(\alpha) = max(ord(\alpha_1), ..., ord(\alpha_k), ord(\beta)) + 1$. 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 in the following way:

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 $\beta, \alpha_i \in Types$, $i \neq j \Rightarrow x_i \neq x_j, i, j = 1, ..., k, k \geq 1$ 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 occurrence of free variable in the term is called internal if it does not enter the applicator, the scope of which contains a free occurrence of some variable. The occurrence of free variable in the term is called external if it does not enter the scope of the applicator that contains a free occurrence of some variable.

Let $t \in \Lambda_\alpha, \alpha \in Types$ and $FV(t) \subset \{y_1, ..., y_n\}, \overline{y}_0 = (y_1^0, ..., y_n^0)$, 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 $\overline{y}_0 = (y_1^0, ..., y_n^0)$, is denoted by $Val_{\overline{y}_0}(t)$ 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 $\overline{y}_0 = (y_1^0, ..., y_n^0)$, where $y_i^0 \in V_{\beta_i}, i = 1, ..., n$ we have the following: $Val_{\overline{y}_0}(t_1) = Val_{\overline{y}_0}(t_2)$. 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 the considered terms use variables of any order and constants of order ≤ 1, where constants of order 1 are strong 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 strong computable if there exists an algorithm, which stops with the value $f(m_1, ..., m_k) \in M$ for all $m_1, ..., m_k \in M$, see [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 of 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 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 $f \in [M^k \to M]$, $t_i \in \Lambda_M, i = 1, ..., k, k \geq 1$, 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 ($\rightarrow_{\beta\delta}$) and $\beta\delta$-reduction ($\rightarrow\!\!\rightarrow_{\beta\delta}$) are defined in the conventional way. A term containing no $\beta\delta$-redexes is called a 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 $f \in [M^k \rightarrow M]$, $t_i \in \Lambda_M, i = 1, ..., k, k \geq 1$, 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. $t \in \beta\text{-}NF, t \sim m, m \in M \setminus \{\perp\} \Rightarrow t \rightarrow\!\!\rightarrow {}_{\delta} m$,
2. $t \in \beta\text{-}NF, FV(t) = \emptyset, t \sim \perp \Rightarrow t \rightarrow\!\!\rightarrow_{\delta} \perp$.

Theorem 1. *(on canonical notion of δ-reduction). For every recursive set of strong computable, monotonic functions with indeterminate values of arguments there exists a canonical notion of δ-reduction.*

Proved in [3].

Typed functional program P is the following system of equations:

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

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

Every typed functional program P has the least solution (see [1]). Let $(f_1, ..., f_n) \in \alpha_1 \times ... \times \alpha_n$ be the least solution of P, then the first component $f_1 \in [M^k \rightarrow M]$ of the least solution is said to be the basic semantics of the program P and is denoted by f_p.

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

2. Interpretation Algorithms, $\perp$-Dependence

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. Algorithm A stops with the result $m \in M$ or works infinitely. 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.

$Proc_A(P) = \{(m_1, ..., m_k, m) |$ algorithm A stops for the program P and the term $F_1(m_1, ..., m_k)$ with the 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 2. *Every interpretation algorithm A is consistent.*

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

Definition 2. *An interpretation algorithm A $\perp$-depends on canonical notion of δ-reduction if there exist δ_1 and δ_2 canonical notions of δ-reduction, program P and $m_1, ..., m_k \in M, k \geq 1$ such that: $(m_1, ..., m_k, \perp) \in Proc_A(P)$ for δ_1 and $(m_1, ..., m_k, \perp) \notin Proc_A(P)$ for δ_2.*

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 3. *(Interpretation algorithms) FS, PES, LES, PIS, LIS, PAS, ACT*

STEP 1:

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

STEP 2:

FS, PES, LES, PIS, LIS: If $t \equiv t_\tau$ where τ is a 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 $A \in \{ACT, PAS\}$, 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 $PES(P, t < t_{i_1}, ..., t_{i_k} >)$.

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 $LES(P, t_{t_i})$.

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 $PIS(P, t < t_{i_1}, ..., t_{i_k} >)$.

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 $LIS(P, t_{t_i})$.

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 a leftmost redex, then $ACT(P, t_{\tau[ACT(P,\tau_1),...,ACT(P,\tau_r)]})$, 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 a leftmost redex, then $PAS(P, t_{\tau[\tau_1,...,\tau_r]})$, else go to Step 4.

STEP 4:

ACT, PAS: If $t \equiv t_\tau$ where τ is a 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 3. *Any interpretation algorithm FS, PES, LES, PIS, LIS, PAS, ACT $\perp$-depends on canonical notion of δ-reduction.*

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

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

It is easy to see that not_eq is a strong computable, naturally extended function with indeterminate values of arguments (a function is said to be naturally extended, if its value is $\perp$ whenever the value of at least one of the arguments is $\perp$). Therefore, from Theorem 1 it follows that there exists the following canonical notion 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 $n_1 \neq n_2$
$(not\_eq(n, n), \perp) \in \delta$, where $n \in N$
$(not\_eq(n, \perp), \perp) \in \delta$, where $n \in N$
$(not\_eq(\perp, n), \perp) \in \delta$, where $n \in N$
$(not\_eq(\perp, \perp), \perp) \in \delta$

Let us define two canonical notions of δ-reduction δ_1 and δ_2.

δ_1 is: $(not\_eq(n_1, n_2), 1) \in \delta_1$, where $n_1, n_2 \in N$ and $n_1 \neq n_2$
$(not\_eq(t, t), \perp) \in \delta_1$, where $t \in \Lambda_M$
$(not\_eq(t, \perp), \perp) \in \delta_1$, where $t \in \Lambda_M$
$(not\_eq(\perp, t), \perp) \in \delta_1$, where $t \in \Lambda_M$

δ_2 is: $(not\_eq(n_1, n_2), 1) \in \delta_2$, where $n_1, n_2 \in N$ and $n_1 \neq n_2$
$(not\_eq(m, m), \perp) \in \delta_2$, where $m \in M$
$(not\_eq(t, \perp), \perp) \in \delta_2$, where $t \in \Lambda_M$
$(not\_eq(\perp, t), \perp) \in \delta_2$, where $t \in \Lambda_M$

It is easy to see that δ_1 is an effective, single-valued notion of δ-reduction. Therefore, to show that δ_1 is a canonical notion of δ-reduction it suffices to show that $\delta \subset \delta_1$. Let $(\tau_1, \tau_2) \in \delta$, where $\tau_1, \tau_2 \in M$, then the following cases are possible:

$\tau_1 \equiv n_1$ and $\tau_2 \equiv n_2$, where $n_1, n_2 \in N$ and $n_1 \neq n_2$, then it is obvious that $(not\_eq(n_1, n_2), 1) \in \delta_1$.
$\tau_1 \equiv \tau_2 \equiv n$, where $n \in N$, then from $(not\_eq(t, t), \perp) \in \delta_1$, where $t \in \Lambda_M$, follows that $(not\_eq(n, n), \perp) \in \delta_1$.
$\tau_1 \equiv n$ and $\tau_2 \equiv \perp$, where $n \in N$, then from $(not\_eq(t, \perp), \perp) \in \delta_1$, where $t \in \Lambda_M$, follows that $(not\_eq(n, \perp), \perp) \in \delta_1$.
$\tau_1 \equiv \perp$ and $\tau_2 \equiv n$, where $n \in N$, then from $(not\_eq(\perp, t), \perp) \in \delta_1$, where $t \in \Lambda_M$, follows that $(not\_eq(\perp, n), \perp) \in \delta_1$.
$\tau_1 \equiv \perp$ and $\tau_2 \equiv \perp$, then from $(not\_eq(t, t), \perp) \in \delta_1$, where $t \in \Lambda_M$, follows that $(not\_eq(\perp, \perp), \perp) \in \delta_1$.

It is easy to see that δ_2 is an effective, single-valued notion of δ-reduction. Therefore, to show that δ_2 is a canonical notion of δ-reduction it suffices to show that $\delta \subset \delta_2$. Let $(\tau_1, \tau_2) \in \delta$, where $\tau_1, \tau_2 \in M$, then the following cases are possible:

$\tau_1 \equiv n_1$ and $\tau_2 \equiv n_2$, where $n_1, n_2 \in N$ and $n_1 \neq n_2$, then it is obvious that $(not\_eq(n_1, n_2), 1) \in \delta_2$.
$\tau_1 \equiv \tau_2 \equiv n$, where $n \in N$, then from $(not\_eq(m, m), \perp) \in \delta_2$, where $m \in M$, follows that $(not\_eq(n, n), \perp) \in \delta_2$.
$\tau_1 \equiv n$ and $\tau_2 \equiv \perp$, where $n \in N$, then from $(not\_eq(t, \perp), \perp) \in \delta_2$, where $t \in \Lambda_M$, follows that $(not\_eq(n, \perp), \perp) \in \delta_2$.

$\tau_1 \equiv \perp$ and $\tau_2 \equiv n$, where $n \in N$, then from $(not\_eq(\perp, t), \perp) \in \delta_2$, where $t \in \Lambda_M$, follows that $(not\_eq(\perp, n), \perp) \in \delta_2$.

$\tau_1 \equiv \perp$ and $\tau_2 \equiv \perp$, then from $(not\_eq(m, m), \perp) \in \delta_2$, where $m \in M$, follows that $(not\_eq(\perp, \perp), \perp) \in \delta_2$.

Let P be the following program, where $F_1, F_2 \in V_{[M \to M]}, x \in V_M$

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

For δ_1, program P and $FS, PES, LES, PIS, LIS, PAS, ACT$ we have:
$F_1(0)$;
$\lambda x[not\_eq(F_2(x), F_2(x))](0) \to_\beta not\_eq(F_2(0), F_2(0)) \to_{\delta_1} \perp$;
Therefore $(0, \perp) \in Proc_{FS}(P)$, $(0, \perp) \in Proc_{PES}(P)$, $(0, \perp) \in Proc_{LES}(P)$, $(0, \perp) \in Proc_{PIS}(P)$, $(0, \perp) \in Proc_{LIS}(P)$, $(0, \perp) \in Proc_{PAS}(P)$, $(0, \perp) \in Proc_{ACT}(P)$.

For δ_2, program P and ACT, LIS, PAS, LES we have:
$F_1(0)$;
$\lambda x[not\_eq(F_2(x), F_2(x))](0) \to_\beta not\_eq(F_2(0), F_2(0))$;
$not\_eq(\lambda x[F_2(x)](0), F_2(0)) \to_\beta not\_eq(F_2(0), F_2(0))$;
... and so on.
Therefore $(0, \perp) \notin Proc_{LES}(P)$, $(0, \perp) \notin Proc_{LIS}(P)$, $(0, \perp) \notin Proc_{PAS}(P)$, $(0, \perp) \notin Proc_{ACT}(P)$.

For δ_2, program P and FS, PES, PIS we have:
$F_1(0)$;
$\lambda x[not\_eq(F_2(x), F_2(x))](0) \to_\beta not\_eq(F_2(0), F_2(0))$;
$not\_eq(\lambda x[F_2(x)](0), \lambda x[F_2(x)](0)) \to_\beta not\_eq(F_2(0), F_2(0)))$;
... and so on.
Therefore $(0, \perp) \notin Proc_{FS}(P)$, $(0, \perp) \notin Proc_{PES}(P)$, $(0, \perp) \notin Proc_{PIS}(P)$.
In conclusion, for each interpretation algorithm $A \in \{FS, PES, LES, PIS, LIS, PAS, ACT\}$ there exist δ_1 and δ_2 canoncial notions of δ-reduction and program P such that $(0, \perp) \in Proc_A(P)$ for δ_1 and $(0, \perp) \notin Proc_A(P)$ for δ_2, therefore A $\perp$-depends on canonical notion of δ-reduction. ■

References

[1] S. A. Nigiyan "Functional Languages", *Programming and Computer Software*, vol. 17, no. 5, pp. 290-297, 1992.

[2] S. A. Nigiyan, "On non-classical theory of computability", *Proceedings of the YSU, Physical and Mathematical Sciences*, no.1, pp.52-60, 2015.

[3] S. A. Nigiyan and T.V.Khondkaryan, "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*, no. 1, pp. 46-52, 2017.

[4] R. Yu. Hakopian, "On procedural semantics of strong typed functional programs", *Proceedings of YSU, Natural Sciences*, (in Russian), no. 3, pp.59-69, 2008.

Submitted 10.10.2017, accepted 18.01.2018.

Տիպիզացված ֆունկցիոնալ ծրագրերի ինտերպրետացիայի ալգորիթմների կախվածությունը կանոնիկ δ-ռեդուկցիայի գաղափարից

Դ. Գրիգորյան

Ամփոփում

Աշխատանքում դիտարկված են տիպիզացված ֆունկցիոնալ ծրագրերի ինտերպրետացիայի ալգորիթմները: Ինտերպրետացիայի ալգորիթմը հիմնված է տեղադրման, β-ռեդուկցիայի և կանոնիկ δ-ռեդուկցիայի գործողությունների վրա: Տիպիզացված ֆունկցիոնալ ծրագրերի հիմնական սեմանտիկան անորոշ արգումենտներով ֆունկցիա է, որը փոքրագույն լուծման հիմնական բաղադրիչն է: Եթե հիմնական սեմանտիկայի արժեքը որոշ արժեքների դեպքում անորոշ է, ապա ինտերպրետացիայի ալգորիթմը կամ կանգ է առնում $\perp$ արժեքով, կամ աշխատում է անվերջ: Ցույց է տրված, որ 7 հայտնի ինտերպրետացիայի ալգորիթմները $\perp$-կախված են կանոնիկ δ-ռեդուկցիայի գաղափարից: Այդ ալգորիթմները հետևյալն են՝ FS (լիրվ տեղադրման), PES (զուգահեռ արտաքին տեղադրման), LES (ձախ արտաքին տեղադրման), PIS (զուգահեռ ներքին տեղադրման), LIS (ձախ ներքին տեղադրման), ACT (ակտիվ ալգորիթմ), PAS (պասիվ ալգորիթմ).

О зависимости алгоритмов интерпретации типизированных функциональных программ от канонического понятия δ-редукции

Д. Григорян

Аннотация

В данной работе рассматриваются интерпретаторы типизированных функциональных программ. Алгоритм интерпретации основан на подстановках, β-редукции и канонической δ-редукции. Основная семантика типизированных функциональных программ есть функция с неопределенными значениями аргументов, которая является главной компонентой ее наименьшего решения. Если значение основной семантики, для некоторых значений аргументов, есть неопределенность, то алгоритм интерпретации либо останавливается со значением $\perp$, либо работает бесконечно. Показано, что семь известных алгоритмов интерпретации $\perp$-зависят от канонического понятия δ-редукции. Вот эти алгоритмы: FS (полной подстановки), PES (параллельной внешней подстановки), LES (левой внешней подстановки), PIS (параллельной внутренней подстановки), LIS (левой внутренней подстановки), ACT (активный алгоритм), PAS (пассивный алгоритм).