PROCEEDINGS OF THE YEREVAN STATE UNIVERSITY

Physical and Mathematical Sciences 2011, № 1, p. 36–43

Informatics

ON TRANSFORMATIONS OF UNTYPED FUNCTIONAL PROGRAMS AND THEIR PROCEDURAL SEMANTICS

G. A. GHAZARYAN*

Chair of System Programming, RAU

In this paper the notion of transformation of untyped functional programs that preserves the main semantics of programs is presented. A transformation, representing programs by two equations, such that procedural semantics of programs that use interpretation algorithms based on substitution and normal form reduction remain the same, is introduced. It is proved also that there is no transformation, which represents programs with one equation, such that the procedural semantics of programs remain unchanged for all interpretation algorithms that are based on two operations: a substitution and a one-step β-reduction.

***Keywords*:** term, equation, procedural semantics, interpretation algorithms.

1. Introduction. In the present paper the fixed-point (main) semantics of untyped functional programs and procedural semantics that use interpretation algorithms, on which the interpreters of untyped functional programming systems are based (or can be based), are considered. These algorithms use two operations: a substitution and a one-step β-reduction. The notion of transformation of untyped functional programs that preserves the main semantics is presented. We introduce the transformation, which represents programs by two equations, such that procedural semantics using the interpretation algorithms based on substitution and normal form reduction remains unchanged. It is proved also that there is no transformation that represents programs by one equation, such that the procedural semantics remains unchanged the same for all interpretation algorithms.

2. Definitions Used and Previous Results. Main definitions and notations used in this paper are borrowed from [1, 2]. Let V be a countable set of variables.

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

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

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

* E-mail: gghazaryan@hotmail.com

The notions of free and bound occurrence of a variable in the term and the notion of a free variable of the term are introduced in the conventional way. The set of all free variables of the term t is denoted as $fv(t)$. A term that does not contain free variables is called closed.

To show mutually different variables of interest $x_1, x_2, ..., x_n, n \geq 1$, of the term t, the notation $t[x_1, x_2, ..., x_n]$ is used. The notation $t[t_1, t_2, ...t_n]$ (or $t[x_1 := t_1, x_2 := t_2, ..., x_n := t_n]$) denotes the term, obtained by simultaneous substitution of the terms $t_1, t_2, ...t_n$ for all free occurrences of the variables $x_1, x_2, ..., x_n$ respectively, into the term t. The notation $t\langle x_{i_1}, ..., x_{i_k} \rangle$ is used to denote the term t with indication of some $k \geq 0$ free occurrencies of variables $x_1, x_2, ..., x_n$ (from left to right), $x_{i_j} \in \{x_1, x_2, ..., x_n\}, j = 1, ..., k.$ The term, obtained from a term $t\left\langle x_{i_1}, ..., x_{i_k} \right\rangle$ as a result of simultaneous substitution of terms $t_{i_1}, ..., t_{i_k}$ for occurrencies of the variables $x_{i_1}, ..., x_{i_k}$, is denoted as $t\left\langle t_{i_1}, ..., t_{i_k} \right\rangle$.

A substitution is said to be admissible, if all free variables of the term being substituted remain free after substitution. We will consider only admissible substitutions. Terms t_1 and t_2 are said to be congruent (which is denoted as $t_1 \equiv t_2$), if one term can be obtained from the other by renaming the bound variables. The congruent terms are considered to be identical.

The notion of β-reduction is the following:

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

A one-step β-reduction ($\rightarrow_\beta$), β-reduction ($\rightarrow\rightarrow_\beta$) and β-equality ($=_\beta$) are defined in the regular manner. In what follows, we will omit symbol β. The term $(\lambda x.t[x])t'$ is called a β-redex (simply redex). A term not containing redexes is called a β-normal form (simply normal form). We will denote the set of all normal forms by NF and the set of all closed normal forms by NF^0. A term t is said to have a normal form, if there exists a term $t' \in NF$ such that $t = t'$. Also the following notations are introduced:

$T \equiv \lambda xy.x,\ F \equiv \lambda xy.y,\ I \equiv \lambda x.x,$ where $x, y \in V$;

$< t_1, ..., t_m > \equiv \lambda x.xt_1...t_m$, where $t_i \in \Lambda, x \in V, x \notin fv(t_i)$, $i = 1, ..., m,\ \ m \geq 1$;

$U_i^m \equiv \lambda x_1...x_m.x_i$, where $x_i \in V,\ k \neq j \Rightarrow x_k \neq x_j,\ k, j = 1, ..., m,\ 1 \leq i \leq m, m \geq 1$;

$P_i^m \equiv \lambda x.xU_i^m$, where $x \in V,\ 1 \leq i \leq m,\ \ m \geq 1$;

$Y \equiv \lambda h.\left(\lambda x.h(xx)\right)\left(\lambda x.h(xx)\right)$ is a fixed-point combinator, where $x, h \in V$.

The following definitions are used from [2].

Definition 2.2. An untyped functional program (simply program) is a system of equations of the form

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

where $f_i \in V,\ i \neq j \Rightarrow f_i \neq f_j,\ t_i[f_1,...,f_m] \in \Lambda,\ fv(t_i[f_1,...f_m]) \subseteq \{f_1,...,f_m\},\ i,j = 1,...,m,$ $m \geq 1$. The first equation of the system is considered to be the principal equation of the program. Let us consder the solution of system (1)

$$(\tau_1,...,\tau_m), \tag{2}$$

where $\tau_i \equiv P_i^m\left(Y\left(\lambda x. < t_1[P_1^m x,...,P_m{}^m x],...,t_m[P_1^m x,...,P_m{}^m x] >\right)\right),\ i = 1,...,m.$ The term τ_1 is said to be the principal component of solution (2); it is this term that is the fixed-point semantics of program (1).

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

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

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

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

b) a one-step β -reduction.

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

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

Let $L(X)$ be the term X, if $X \in NF$ and X' be obtained from X by applying a left one-step reduction. We define algorithm N that, by a given term X, constructs its normal form, if it exits or functions endlessly if not.

Algorithm N.

Input: term X.

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

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

Let us describe the set of interpretation algorithms $SNFR$, in which the reduction to the normal form alternates with the substitution operation.

Each algorithm $A \in SNFR$ has the following form:

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

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

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

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

In what follows, only the algorithms $A \in SNFR$ satisfying to the following condition (*) are considered:

if $t\langle f\rangle \in NF \setminus NF^0$, $f \in fv(t)$ and $t' \equiv t\langle\tau\rangle$, $\tau \in NF$ and τ contains only one free occurrence of a variable, then algorithm A selects the same free occurrences of the variables for terms t and t' in step 2.

Algorithm ACT.

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

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

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

2. If $X \equiv X\langle f_i\rangle$, where f_i is the leftmost occurrence of variables $\{f_1,...,f_m\}$ in the term t and this particular occurrence is on the left of the leftmost redex of the term t, then $ACT(P, t\langle t_i\rangle)$; else go to 3.

3. If $X \equiv X_{(\lambda x.t)\tau}$, where $x \in V$, $t, \tau \in \Lambda$ and $(\lambda x.t)\tau$ is the leftmost redex of term X, then $ACT(P, X_{t[x:=ACT(P,\tau)]})$.

3. Transformations of Functional Programs and their Procedural Semantics.

We say that procedural semantics using interpretation algorithm A is consistent (complete), if $\mathrm{Proc}_A(P) \subseteq Fix(P)$ ($Fix(P) \subseteq \mathrm{Proc}_A(P)$ respectively) for any program P.

The following theorems are used from [2].

Theorem 3.1 *(on consistency).* For any program P and interpretation algorithm A $\mathrm{Proc}_A(P) \subseteq Fix(P)$.

Theorem 3.2 *(on substitutions).* Let P be a program of form (1) and $v_1,...,v_k,t_0 \in NF^0$, where $k \geq 0$, then $(v_1,...,v_k,t_0) \in Fix(P) \Leftrightarrow \exists\, n \geq 1$; $t_1^n[f_1,...,f_m]v_1...v_k \to\to t_0$, where $t_i^0[f_1,...,f_m] \equiv f_i$,

$t_i^s[f_1,...,f_m] \equiv t_i[t_1^{s-1}[f_1,...,f_m],...,t_m^{s-1}[f_1,...,f_m]]$, $s \geq 1$, $i = 1,...,m$.

We will denote the set of all untyped functional programs by P.

Definition 3.1. The mapping $T: \mathsf{P} \to \mathsf{P}$ is called the transformation of programs P, if for any program $P \in \mathsf{P}$ $Fix(T(P)) = Fix(P)$.

Theorem 3.3. There does not exist such a transformation T that for any program $P, T(P)$ is composed of one equation and satisfies the following condition:

$$\mathrm{Proc}_A(P) = \mathrm{Proc}_A(T(P))$$

for any interpretation algorithm A.

For the proof of Theorem 3.3 we will use Lemma 3.1.

Lemma 3.1. Let the program P be $f = t$, where $t \in \Lambda, fv(t) = \{f\}$. Then $\mathrm{Proc}_{ACT}(P) = \varnothing$.

Proof of Lemma 3.1. Let us suppose the opposite: there exists a program P, which satisfies to the conditions of the Lemma 3.1 and $\mathrm{Proc}_{ACT}(P) \neq \varnothing$. By Definition 2.4, $\mathrm{Proc}_{ACT}(P) \neq \varnothing \Rightarrow \exists v_1, v_2,..., v_k, t_0 \in NF^0$; $ACT(P, tv_1...v_k) = t_0$, $k \geq 0$.

We will show that algorithm ACT executes endlessly on the term $tv_1...v_k$. Since $fv(tv_1...v_k) = \{f\}$ and $t_0 \in NF^0$, consequently, all free occurences of variable f will be removed at a certain step of the algorithm. As $fv(t) = \{f\}$, then the variable f cannot be removed by replacing its free occurences with t. From there, a free occurence of variable f can be removed only after the third step of algorithm ACT. Let us suppose that at the third step the algorithm ACT has selected the redex $(\lambda x.t)\tau$. To remove a free occurence of variable f it is required that the following condition be satisfied: $f \in fv(\tau)$. As it follows from the third step of the algorithm ACT, it should calculate $ACT(P,\tau)$. It becomes evident that the same logical steps done above for the term $tv_1...v_k$ can be repeated for the term τ. Hence, the algorithm ACT executes endlessly contrary to the fact that $ACT(P,tv_1...v_k) = t_0$.

Lemma 3.1 is proved.

Proof of Theorem 3.3. Let us suppose the opposite: there exists such a transformation T, which satisfies to the conditions of Theorem 3.3. In what follows, we refer to this as a main assumption. Let us consider the program P:

$f_1 = \lambda x.(\lambda x.f_2(xx))(\lambda x.f_2(xx)) \equiv t_1$,

$f_2 = \lambda x.I \equiv t_2$,

where $f_1, f_2, x \in V$.

We will show that $ACT(P,t_1[f_1,f_2]I) \equiv I$. Let us write the steps of the execution of algorithm ACT on the program P and on the term $t_1[f_1,f_2]I$:

$ACT(P,(\lambda x.f_2(xx))(\lambda x.f_2(xx))I)$,
$ACT(P,(\lambda x.f_2(xx))(\lambda x.f_2(xx))[x := ACT(P,I)])$,
$ACT(P,(\lambda x.f_2(xx))(\lambda x.f_2(xx))[x := I])$,
$ACT(P,(\lambda x.f_2(xx))(\lambda x.f_2(xx)))$,
$ACT(P,f_2(xx)[x := ACT(P,\lambda x.f_2(xx))])$,
$ACT(P,f_2(xx)[x := ACT(P,\lambda x.(\lambda x.I)(xx))])$,
$ACT(P,f_2(xx)[x := ACT(P,\lambda x.I[x := ACT(P,xx)])])$,
$ACT(P,f_2(xx)[x := ACT(P,\lambda x.I[x := xx])])$,
$ACT(P,f_2(xx)[x := ACT(P,\lambda x.I)])$,
$ACT(P,f_2(xx)[x := \lambda x.I])$,
$ACT(P,f_2((\lambda x.I)(\lambda x.I)))$,
$ACT(P,(\lambda x.I)((\lambda x.I)(\lambda x.I)))$,
$ACT(P,I[x := ACT(P,(\lambda x.I)(\lambda x.I))])$,
$ACT(P,I[x := ACT(P,I[x := ACT(P,\lambda x.I)])])$,
$ACT(P,I[x := ACT(P,I[x := \lambda x.I])])$,
$ACT(P,I[x := ACT(P,I)])$,
$ACT(P,I[x := I])$,
$ACT(P,I)$,
I.

Hence, $(I,I) \in \mathrm{Proc}_{ACT}(P) \Rightarrow \mathrm{Proc}_{ACT}(P) \neq \varnothing$.

According to Theorem 3.1 of consistency, $(I,I) \in Fix(P)$.

It is easy to see that as the term $t_1[f_1, f_2]I$ has not a normal form, hence, $(I,I) \notin \mathrm{Proc}_A(P) \Rightarrow \mathrm{Proc}_A(P) \neq Fix(P)$ for any interpretation algorithm $A \in SNFR$.

Now, let us suppose $T(P) = P_0$ and the program P_0 is composed of the following equation: $f_0 = t_0$, where $f_0 \in V, t_0 \in \Lambda$.

There are two possible cases:

1) $f_0 \in fv(t_0)$ – in this case, by Lemma 3.1, $\mathrm{Proc}_{ACT}(P_0) = \varnothing$. And because $\mathrm{Proc}_{ACT}(P) = \varnothing$, consequently, this contradicts our main assumption.

2) $f_0 \notin fv(t_0)$ – in this case, according to Theorem 3.2, $\mathrm{Proc}_A(P_0) = Fix(P_0)$ for any interpretation algorithm $A \in SNFR$.

As we have already shown $\mathrm{Proc}_A(P) = Fix(P)$ for any interpretation algorithm $A \in SNFR$ and by the definition of transformation T, $Fix(P) = Fix(P_0)$. Therefore, we obtain that $\mathrm{Proc}_A(P_0) \neq \mathrm{Proc}_A(P)$ for any interpretation algorithm $A \in SNFR$.

The last result contradicts our main assumption. Hence, such a transformation T does not exist.

Theorem 3.3 is proved.

Let us introduce some notations and notations to be used below.

We will denote the term $t\underbrace{F\ldots F}_{n}T$ by $t^{[n]}$, where $t \in \Lambda$, $n \geq 0$.

Let the program P of form (1) be given and $t \in \Lambda$, $fv(t) \subseteq \{f_1,\ldots,f_m\}$. We use the notation $\overline{t}$ to denote the term obtained from t by simultaneous substitution of terms $f_2^{[j-1]}, j = 1,\ldots,m$, for all free occurences of the variables f_j, $j = 1,\ldots,m$, respectively.

Algorithm C.

Input: terms $t_1,\ldots,t_m, m \geq 1$.

Output: term $C(t_1,\ldots,t_m)$.

1. If $m = 1$, then $\lambda x.xt_1I$; else $\lambda x.xt_1C(t_2,\ldots,t_m)$.

Algorithm Converter.

Input: program P of form (1).

Output: program *Converter*(P).

1. Return the following program:

$$\begin{aligned} f_1 &= t_1'[f_2] \equiv f_2T, \\ f_2 &= t_2'[f_2] \equiv \overline{C(t_1,\ldots,t_m)}. \end{aligned} \tag{3}$$

Theorem 3.4. For any program P of form (1) $Fix(Converter(P)) = Fix(P)$.

For the proof of Theorem 3.4 the Proposition 3.1 is used.

Proposition 3.1. Term t has a closed normal form $N(t)$, iff $\overline{t}$ has a closed normal form $N(\overline{t})$ and $N(t) \equiv N(\overline{t})$.

Proof of Theorem 3.4. It is easy to see that the definition of $t_i^n, i = 1,\ldots,m$,

which is given in the Theorem 3.2, can be also intoduced in the following way:

$t_i^0[f_1,...,f_m] \equiv f_i$,

$t_i^s[f_1,...,f_m] \equiv t_i^{s-1}[t_1,...,t_m]$, $s \geq 1$, $i = 1,...,m$.

Now suppose that $Converter(P)$ is the program (3).

By definition of $Converter(P)$, it is easy to see that:

1. $C(t_1,...,t_m)^{[i-1]} \to\to t_i$, $i = 1,...,m$;
2. $\overline{C(t_1,...,t_m)}^{[i-1]} \to\to \overline{t_i}$, $i = 1,...,m$.

Now let us show that $Fix(P) = Fix(Converter(P))$. For this reason, according to Theorem 3.2, it is sufficient to show that for any $n \geq 1$ and for any terms $v_1,...,v_k$, $t_0 \in NF^0$, $t_1'^{n+1}v_1...v_k \to\to t_0 \Leftrightarrow t_1^n v_1...v_k \to\to t_0$.

Using the method of mathematical induction we will prove that $t_1'^{n+1} \to\to \overline{t_1^n}$, $n \geq 1$.

Basis: Let us show that $t_1'^2 \to\to \overline{t_1^1}$.

Since $t_1'^2 \equiv t_1'[t_2'] \equiv t_2'T \equiv \overline{C(t_1,...,t_m)}^{[0]}$ and $\overline{C(t_1,...,t_m)}^{[0]} \to\to \overline{t_1}$, then $t_1'^2 \to\to \overline{t_1^1}$.

Induction hypothesis: Let $n \geq 2$ and $t_1'^n \to\to \overline{t_1^{n-1}}$.

Induction step: We will show that $t_1'^{n+1} \to\to \overline{t_1^n}$.

By definition of t_1^n, $t_1^n \equiv t_1^{n-1}[t_1,...,t_m]$ and $t_1'^{n+1} \equiv t_1'^n[t_2']$. From the induction hypothesis it follows $t_1'^n \to\to \overline{t_1^{n-1}}$. If $fv(t_1^{n-1}) = \varnothing$, then it becomes evident that $t_1'^{n+1} \to\to \overline{t_1^n}$.

Let us assume that f_j, $j = 1,...,m$, is the k^{th} free occurrence of the variables $\{f_1,...,f_m\}$ in the term t_1^{n-1}: $t_1^{n-1} < f_j >$, $k \geq 1$.

Let us consider the k^{th} free occurrence of f_2 in the term $\overline{t_1^{n-1}}$: $\overline{t_1^{n-1}} < f_2 >$. It is easy to see that the term $\overline{t_1^{n-1} < t_j >}$ is obtained from the term $\overline{t_1^{n-1} < t_2' >}$ by replacing the subterm $\overline{C(t_1,...,t_m)}^{[j-1]}$ with the term $\overline{t_j}$. Hence, $\overline{t_1^{n-1} < t_2' >} \to\to \overline{t_1^{n-1} < t_j >}$.

As the last result is true for any k^{th} free occurrence $k \geq 1$, consequently, $\overline{t_1^{n-1}}[t_2'] \to\to \overline{t_1^{n-1}[t_1,...,t_m]}$. Since $t_1'^n[t_2'] \to\to \overline{t_1^{n-1}}[t_2']$, then $t_1'^n[t_2'] \to\to \overline{t_1^{n-1}[t_1,...,t_m]}$ and $t_1'^{n+1} \to\to \overline{t_1^n}$. Hence, $t_1'^{n+1} = \overline{t_1^n}$.

From here it becomes evident that the term $t_1'^{n+1}v_1...v_k$ has a closed normal form $N(t_1'^{n+1}v_1...v_k)$, iff the term $\overline{t_1^n}v_1...v_k$ has a closed normal form $N(\overline{t_1^n}v_1...v)$ and $N(t_1'^{n+1}v_1...v_k) \equiv N(\overline{t_1^n}v_1...v_k) \equiv t_0$.

Hence, from Proposition 3.1 it follows that $t_1'^{n+1}v_1...v_k \to\to t_0 \Leftrightarrow t_1^n v_1...v_k \to\to t_0$.

Theorem 3.4 is proved.

Now introduce the notation of $A_1^k(P, t_0)$, which is used below.

Let us consider the k^{th}, $k \geq 1$, iteration of the execution of interpretation algorithm A on program P and term t_0. We will assume that algorithm A calculates $N(X)$ in step 1, where the term X is obtained from t_0 by $k-1$ executions of step 1 and step 2 of interpretation algorithm A. If $N(X)$ is not determined, then $A_1^k(P, t_0)$ is not determined, otherwise, $A_1^k(P, t_0) \equiv N(X)$.

Let the program P of form (1) is given and $t \in \Lambda, fv(t) \subseteq \{f_1, ..., f_m\}$.

Taking into consideration the fact that by condition (*) any interpretation algorithm $A \in SNFR$ selects the same free occurrences of variables for terms t and $\overline{t}$ in step 2, then the following proposition can be proved:

Proposition 3.2. Let the program P of form (1) be given and $Converter(P)$ be the program (3) and $v_1, ..., v_k \in NF^0$. For any $k \geq 1$, $A_1^k(P, t_1 v_1 ... v_k)$ is determined, iff $A_1^{k+1}(Converter(P), t_1' v_1 ... v_k)$ is determined and $A_1^{k+1}(Converter(P), t_1' v_1 ... v_k) \equiv \overline{A_1^k(P, t_1 v_1 ... v_k)}$.

Theorem 3.5. For any program P of form (1) $\text{Proc}_A(Converter(P)) = \text{Proc}_A(P)$ for any interpretation algorithm $A \in SNFR$.

Proof of Theorem 3.5. Let $Converter(P)$ is the program (3).

It is easy to see, that if $(v_1, ..., v_k, t_0) \in \text{Proc}_A(P)$ or $(v_1, ..., v_k, t_0) \in \text{Proc}_A(Converter(P))$, then there exists $k \geq 1$ such that $A_1^k(P, t_1 v_1 ... v_k) \equiv t_0$ or $A_1^{k+1}(Converter(P), t_1' v_1 ... v_k) \equiv t_0$ respectively. Since the term t_0 is a closed normal form, then, according to Proposition 3.2, $A_1^{k+1}(Converter(P), t_1' v_1 ... v_k) \equiv A_1^k(P, t_1 v_1 ... v_k)$. Hence, $\text{Proc}_A(Converter(P)) = \text{Proc}_A(P)$.

Theorem 3.5 is proved.

Received 09.11.2010

REFERENCES

1. **Barendregt H.P.** The Lambda Calculus: Its Syntax and Semantics. Amsterdam, New York, Oxford: North-Holland Pub. Comp., 1981.
2. **Nigiyan S.A., Avetisyan S.A.** Programming and Computer Software, 2002, v. 28, № 3, p. 5–14.

Proc. of the Yerevan State Univ. Phys. and Mathem. Sci., 2011, № 1, p. 36–43.

Գ. Ա. Ղազարյան

Առանց տիպերի ֆունկցիոնալ ծրագրերի ձևափոխությունների և նրանց պրոցեդուրային սեմանտիկաների մասին

Աշխատանքը վերաբերում է առանց տիպերի ֆունկցիոնալ ծրագրերի ձևափոխություններին, որոնք պահպանում են հիմնական սեմանտիկան: Ներկայացվում է ձևափոխություն, որը կամայական ծրագիր ձևափոխում է երկու հավասարումով ծրագրի այնպես, որ պրոցեդուրային սեմանտիկաները, որոնք օգտագործում են տեղադրման և նորմալ ձևի բերման վրա հիմնված ինտերպրետացիայի ալգորիթմներ, մնում են նույնը: Ապացուցվում է նաև, որ գոյություն չունի այնպիսի ձևափոխություն, որ կամայական ծրագիր ձևափոխի մեկ հավասարումով ծրագրի այնպես, որ պրոցեդուրային սեմանտիկաները մնան նույնը տեղադրման և միաքայլ β-ռեդուկցիայի գործողությունների վրա հիմնված ինտերպրետացիայի բոլոր ալգորիթմների համար:

Г. А. Казарян.

О преобразованиях бестиповых функциональных программ и их процедурных семантиках

Работа посвящена преобразованию бестиповых функциональных программ, которые сохраняют основную семантику. Описывается преобразование, которое каждую программу преобразовывает в программу, состоящую из двух уравнений, так, что процедурные семантики, которые используют алгоритмы интерпретации, основанные на подстановке и приведении к нормальной форме, не меняются. Доказывается также, что не существует такого преобразования, которое преобразовывает каждую программу в программу, состоящую из одного уравнения, так, что процедурные семантики не меняются для всех алгоритмов интерпретации, которые основаны на двух операциях – подстановке и одношаговой β-редукции.