PROCEEDINGS OF THE YEREVAN STATE UNIVERSITY

Physical and Mathematical Sciences 2012, № 2, p. 42–49

Informatics

ON MINIMALITY OF ONE SET OF BUILT-IN FUNCTIONS FOR FUNCTIONAL PROGRAMMING LANGUAGES

G. A. MARTIROSYAN*

Chair of Programming and Information Technologies YSU, Armenia

The functional programming language, which uses the set {*car, cdr, cons, atom, eq, if_then_else*} of built-in functions is Turing complete (see [1]). In the present paper the minimality of this set of functions is proved.

Keywords: functional programming language, built-in function, Turing completeness, minimality.

1. Introduction. In [1] it is proved that any functional programming language, which uses {*car*, *cdr*, *cons*, *atom*, *eq*, *if_then_else*} built-in functions is Turing complete. Theorem 3.1 of this paper shows that the set of built-in functions Φ={*car*, *cdr*, *cons*, *atom*, *eq*, *if_then_else*} is minimal for functional programming languages, which use more than two atoms. Theorem 3.2 shows that the function *eq* is representable in a functional programming language, which uses only two atoms and the set $\Phi \setminus \{eq\}$ of built-in functions; this set is minimal for functional programming languages, which use only two atoms and it is the only proper subset of the set Φ, which is minimal for such languages.

2. Definitions and Preliminary Results.

Definition 2.1. Let M be a partially ordered set, which has a least element ⊥, and each element of M is comparable with itself and ⊥ only. Let us define the set *Types*:

1. $M \in Types$;
2. if $\alpha_1,\ldots,\alpha_n$, $\beta \in Types$, then the set of all monotonic mappings from $\alpha_1\times\ldots\times\alpha_n$ into β (denoted by $[\alpha_1\times\ldots\times\alpha_n\to\beta]$) belongs to *Types*.

Definition 2.2. Let $\alpha \in Types$ The order of the type α is a natural number (defined as $ord(\alpha)$), where:

1. if $\alpha = M$, then $ord(a_n) = 0$;
2. if $\alpha = [\alpha_1 \times \ldots \times \alpha_n \to \beta]$, $\alpha_1,\ldots,\alpha_n$, $\beta \in Types$, then $ord([\alpha_1 \times \ldots \times \alpha_n \to \beta]) =$ $= \max(ord(\alpha_1),\ldots,ord(\alpha_n),\ ord(\beta)+1)$.

For each $\alpha \in Types$ we have an α type countable set of variables V_α. Let $\alpha \in Types$, $ord(\alpha)=n$, $n\geq 0$. If $c\in\alpha$, i.e. c is a constant of type α, then $ord(c)=n$. If $x \in V_\alpha$, i.e. x is a variable of type α, then $ord(x)=n$.

* E-mail: gevorg.martirosyan@gmail.com

Let $V = \bigcup_{\alpha \in Types} V_\alpha$ and $\Lambda = \bigcup_{\alpha \in Types} \Lambda_\alpha$, where Λ_α is a set of typed λ-terms of type α. Let us define the set of all terms Λ.

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 $\tau \in \Lambda_{[\alpha_1 \times \ldots \times \alpha_k \to \beta]}, t_i \in \Lambda_\alpha, a_1, \ldots, a_k, \beta \in Types$, $i = 1, \ldots, k$, $k \geq 1$, then $\tau(t_1, \ldots, t_k) \in \Lambda_\beta$.
4. If $\tau \in \Lambda_\beta$, $x_i \in V_\alpha, a_1, \ldots, a_k, \beta \in Types$, $i \neq j \Rightarrow x_i \neq x_j$, $i, j = 1, \ldots, k$, $k \geq 1$, then $\lambda x_1 \ldots x_k[\tau] \in \Lambda_{[\alpha_1 \times \ldots \times \alpha_k \to \beta]}$.

The notions of a free and bound occurrence of a variable in a term and the notation of a free variable of a term are introduced in a conventional way. The set of all free variables of a term t is denoted by $FV(t)$. Terms t_1, 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. Congruent terms are considered identical.

Definition 2.3. A functional program P is a system of equations of the form

$$\begin{cases} F_1 = \tau_1 \\ \quad \cdots \\ F_n = \tau_n \end{cases}, \tag{1}$$

where $F_i \in V_{\alpha_i}$, $i \neq j \Rightarrow F_i \neq F_j$, $\tau_i \in \Lambda_{\alpha_i}$, $\alpha_i \in Types$, $FV(\tau_i) \subset \{F_1, \ldots, F_n\}$, $i, j = 1, \ldots, n$, $n \geq 1$, all used constants have an order ≤ 1, constants of order 1 are computable functions and $\alpha_1 = [M^k \to M]$, $k \geq 1$. In [2] it is proved that any program (1) has a least solution. Let $\langle f_1, \ldots, f_n \rangle \in \alpha_1 \times \ldots \times \alpha_k$ is the least solution of the program P, then $f_p = f_1$ will be the fixpoint semantics of the program P.

We will consider functional programming languages (see [3]), which are defined with the following quadruple $L = (M, C, V, \Lambda(C, V))$, where M is a partially ordered set, which has a least element $\perp$, and each element of M is comparable with itself and $\perp$ only, $C = M \cup \Psi$, Ψ is a set of built-in functions, $\Lambda(C, V)$ is the set of all terms, which are constructed using constants and variables only from the sets C and V. By $\wp(L)$ we will denote the set of programs, for which $F_i \in V$, $\tau_i \in \Lambda(C, V)$, $i = 1, \ldots, n$, $n \geq 1$.

Definition 2.4. We will say that the function $f \in [M^k \to M]$, $k \geq 1$, is representable in the language L, if there exists a program $P \in \wp(L)$ such that $f_p = f$, where f_p is the fixpoint semantics of the program P.

Definition 2.5. The set of built-in functions Ψ is called minimal for the language $L = (M, C, V, \Lambda(C, V))$, where $C = M \cup \Psi$, if for any function $f \in \Psi$, f is not representable in the language $L' = (M, C', V, \Lambda(C', V))$, where $C' = M \cup (\Psi \setminus \{f\})$.

The notions of β and δ reductions are given in [4].

We will use the interpretation algorithm *FS* (full substitution and normal form reduction). The completeness of the interpretation algorithm *FS* follows from [4].

We will consider a finite set of atoms, $Atoms = \{a_1, \ldots, a_n\}$, $n \geq 2$, which contains at least two elements (T, $nil \in Atoms$). T and nil correspond to logical true and false values respectively.

Definition 2.6. We define the set of *S-expressions* as follows:

1. If $t \in Atoms$, then $t \in S\text{-}expressions$;
2. If $t_1,...,t_n \in S\text{-}expressions$ ($n \geq 0$), then $(t_1 ... t_n) \in S\text{-}expressions$.

If $l=(t_1 ... t_n)$, $t_1,...,t_n \in$ *S-expressions* ($n \geq 0$), then l is called a list. In the case n=0, the list is empty and denoted by *nil* (which also corresponds to the logical false value). In the case $n > 0$, t_1 and $(t_2 \, ... \, t_n)$ are correspondingly called the head and the tail of the list l.

Let $M=S\text{-}expressions \cup \{\perp\}$ be a partially ordered set, where $\perp$ is the least element of M, and each element of M is comparable with itself and $\perp$ only.

We will consider the following functions, where *car*, *cdr*, $atom \in [M \to M]$, *cons*, $eq \in [M^2 \to M]$, $if\_then\_else \in [M^3 \to M]$,

$$car(m) = \begin{cases} m_1, \text{ if } m=(m_1 \, ... \, m_k), \ m_1 \in S\text{-}expressions, \ i=1,...,k, \ k \geq 1, \\ \perp, \text{ otherwise;} \end{cases}$$

$$cdr(m) = \begin{cases} nil, \text{ if } m=(m_1), \ m_1 \in S\text{-}expressions, \\ (m_2 \cdots m_k), \text{ if } m=(m_1 \cdots m_k), m_i \in S\text{-}expressions, \ i=1,...,k, \ k>1, \\ \perp, \text{ otherwise;} \end{cases}$$

$$cons(m_0, m) = \begin{cases} (m_0), \text{ if } m_0 \in S\text{-}expressions, \ m=nil, \\ (m_0, m_1 \cdots m_k), \text{ if } m=(m_1 \cdots m_k), m_i \in S\text{-}expressions, \ i=1,...,k, \ k \geq 1, \\ \perp, \text{ otherwise;} \end{cases}$$

$$atom(m) = \begin{cases} T, \text{ if } m \in Atoms, \\ nil, \text{ if } m \notin Atoms, \ m \notin \perp, \\ \perp, \text{ otherwise;} \end{cases} \qquad eq(m_1, m_2) = \begin{cases} T, \text{ if } m_1, \ m_2 \in Atoms, \ m_1 = m_2, \\ nil, \text{ if } m_1, \ m_2 \in Atoms, \ m_1 \neq m_2, \\ \perp, \text{ otherwise;} \end{cases}$$

$$if\_then\_else(m_1, m_2, m_3) = \begin{cases} m_2, \text{ if } m_1 \in S\text{-}expressions, \ m_1 \neq nil, \\ m_3, \text{ if } m_1 = nil, \\ \perp, \text{ otherwise.} \end{cases}$$

3. The Main Results. Let $\Phi = \{car, cdr, cons, atom, eq, if\_then\_else\}$.

Theorem 3.1. The set of built-in functions Φ is minimal for the language $L=(M, C, V, \Lambda(C, V))$, where $C=M \cup \Phi$, which uses more than two atoms.

Theorem 3.2. For the languages, which use only two atoms, we have:

a) the function *eq* is representable in the language $L=(M, C, V, \Lambda(C, V))$, where $C=M \cup (\Phi \setminus \{eq\})$;

b) the set of built-in functions $\Phi \setminus \{eq\}$ is minimal for the language $L=(M, C, V, \Lambda(C, V))$, where $C=M \cup (\Phi \setminus \{eq\})$;

c) for any function $f \in \Phi \setminus \{eq\}$, f is not representable in the language $L=(M, C, V, \Lambda(C, V))$, where $C=M \cup (\Phi \setminus \{f\})$.

The proof of Theorems 3.1 and 3.2 will be deduced from Lemmas 3.1–3.6.

We will consider the following notion of δ-reduction:

1. $\langle f(m_1), m \rangle \in \delta$, where $f \in \{car, cdr, atom\}$, $m_1, m \in M$ and $f(m_1) = m$;
2. $\langle g(m_1, m_2), m \rangle \in \delta$, where $g \in \{cons, eq\}$, $m_1, m_2, m \in M$ and $g(m_1, m_2) = m$;

3. $<$ *if nil then* t_1 *else* t_2, $t_2> \in \delta$, where $t_1, t_2 \in \Lambda_M$;
4. $<$ *if m then* t_1 *else* t_2, $t_1> \in \delta$, where $m \in M$, $m \neq nil$, $m \neq \perp$, $t_1, t_2 \in \Lambda_M$;
5. $<$ *if* $\perp$ *then* t_1 *else* t_2, $\perp> \in \delta$, where $t_1, t_2 \in \Lambda_M$.

In [4] it is given the definition of real notion of δ-reduction. Also from [4] it follows that the defined notion of δ-reduction is real.

To each term $t \in \Lambda_\alpha$, $\alpha \in Types$, we will correspond a set $C^0(t)$, which contains constants of order 0 of the term t:

1. If $t \equiv c$, $c \in M$, then $C^0(t)=\{c\}$. If $t \equiv c$, $c \in \alpha$, $\alpha \in Types$, $ord(\alpha) \neq 0$, then $C^0(t)=\varnothing$;
2. If $t \equiv x$, $x \in V$, then $C^0(t)=\varnothing$;
3. If $t \equiv \tau(t_1,\ldots,t_k) \in \Lambda_\beta, \tau \in \Lambda_{[\alpha_1 \times \ldots \times \alpha_k \to \beta]}, t_i \in \Lambda_{\alpha_i}, \alpha_i, \beta \in Types, i=1,\ldots,k, k \geq 1$, then $C^0(\tau(t_1,\ldots,t_k)) = C^0(\tau) \cup C^0(t_1) \cup \ldots \cup C^0(t_k)$;
4. If $t \equiv \lambda x_1 \ldots x_k[\tau] \in \Lambda_{[\alpha_1 \times \ldots \times \alpha_k \to \beta]}$, $\tau \in \Lambda_\beta$, $x_i \in V_{\alpha_i}$, $\alpha_i, \beta \in Types$, $i=1,\ldots,k, k \geq 1$, $i \neq j \Rightarrow x_i \Rightarrow x_j$, $i, j = 1,\ldots,k$, then $C^0(\lambda x_1 \ldots x_k[\tau]) = C^0(\tau)$.

Let us define the change of underlined $\underline{m}$ by m' in a term t (denoted by $t\{\underline{m} \Rightarrow m'\}$), where $t \in L_\alpha$, $\alpha \in Types$, $m, m' \in M$:

1. If $t \equiv c$, $c \in \alpha$, $\alpha \in Types$, then
   1.1. If $t \equiv m$ and m is underlined, then m';
   1.2. If $t \equiv (s_1 \ldots s_n)$, $s_i \in M$, $n \geq 0$, then $t\{\underline{m} \Rightarrow m'\} \equiv (s_1\{\underline{m} \Rightarrow m'\},\ldots, s_n\{\underline{m} \Rightarrow m'\})$;
   1.3. Otherwise, t;
2. If $t \equiv x$, $x \in V$, then t;
3. If $t \equiv \tau(t_1,\ldots,t_k) \in \Lambda_\beta$, $\tau \in \Lambda_{[\alpha_1 \times \ldots \times \alpha_k \to \beta]}$, $t_i \in \Lambda_{\alpha_i}$, $\alpha_i, \beta \in Types$, $i=1,\ldots,k$, $k \geq 1$, then $\tau(t_1,\ldots,t_k)\{\underline{m} \Rightarrow m'\} \equiv \tau\{\underline{m} \Rightarrow m'\}(t_1\{\underline{m} \Rightarrow m'\},\ldots,t_k\{\underline{m} \Rightarrow m'\})$;
4. If $t \equiv \lambda x_1 \ldots x_k[\tau] \in \Lambda_{[\alpha_1 \times \ldots \times \alpha_k \to \beta]}$, $\tau \in \Lambda_\beta$, $x_i \in V_{\alpha_i}$, $i=1,\ldots,k$, $k \geq 1, a_1,\ldots a_k$, $\beta \in Types$, $i \neq j \Rightarrow x_i \neq x_j$, $i, j=1,\ldots,k$, then $\lambda x_1 \ldots x_k[\tau]\{\underline{m} \Rightarrow m'\} \equiv \lambda x_1 \ldots x_k[\tau\{\underline{m} \Rightarrow m'\}]$.

We will say that term t' is obtained from term t(t, $t' \in \Lambda_\alpha$, $\alpha \in Types$) by changing $\underline{m}_1$ by $m'_1,\ldots,\underline{m}_n$ by m'_n (denoted by $t\{\underline{m}_1 \Rightarrow m'_1,\ldots, \underline{m}_n \Rightarrow m'_n\} \equiv t'$), where $\underline{m}_i$, $m'_i \in M$, $i \neq j \Rightarrow \underline{m}_i \neq \underline{m}_j$, $i, j=1,\ldots,n$, $n \geq 1$, if there exist terms $t_0,\ldots,t_n \in \Lambda_\alpha$ such that $t \equiv t_0$, $t' \equiv t_n$ and $t_i\{\underline{m}_i \Rightarrow m'_i\} \equiv t_{i+1}$, $i=0,\ldots n-1$, $n \geq 1$.

Let us consider the functional programming language
$L_1=(M, C_1, V, \Lambda(C_1, V))$, where $C_1=M \cup (\Phi \setminus \{car\})$.

Lemma 3.1. The function *car* is not representable in the language L_1.

Proof. We will prove this Lemma by contradiction. Let us assume that the function *car* is representable in the language L_1. That means there exists a program $P_1 \in \wp(L_1)$, ($F_1 \in V_{[M \to M]}$) such that $f_{P_1}=car$. We consider the action of the interpretation algorithm *FS* for two cases: $FS(P_1, F_1((T)))$ and $FS(P_1, F_1((nil)))$. $FS(P_1, F_1((T)))$ and $FS(P_1, F_1((nil)))$ are definied, because $car((T))$ and $car((nil))$ are defined and the interpretation algorithm *FS* is complete.

If $FS(P_1, F_1((T))) \neq T$, then $f_{P_1} \neq car$, and we will get a contradiction. So, let us assume that $FS(P_1, F_1((T)))=T$. We will show that $FS(P_1, F_1((nil)))=T \neq nil$, so, $f_{P_1} \neq car$ and we will get a contradiction again.

In the term $F_1((T))$, which is an input data of the interpretation algorithm FS, the atom T will be underlined. So, we will consider $FS(P_1, F_1((\underline{T})))$ and $FS(P_1, F_1((nil)))$.

We will consider two sequences of terms $t_0,t_1,\cdots$ and $t'_0,t'_1,\cdots$. $t_0 \equiv F_1((\underline{T}))$, and for any $i \geq 0$, t_{i+1} is obtained from t_i by applying one step of the interpretation algorithm FS with input data P_1 and t_i. Also let $t'_0 \equiv F_1((nil))$ and for any $i \geq 0$, t'_{i+1} is obtained from t'_i by applying one step of the interpretation algorithm FS with input data P_1 and t'_i.

There exists $n > 0$ such that $t_n = T$, because $FS(P_1, F_1((T)))=T$, and the term t_{i+1} is obtained from t_i by applying one of the steps of the interpretation algorithm FS with input data P_1 and t_i.

By induction it can be proved that for any $0 \leq I \leq n$, $\underline{T} \notin C^0(t_i)$ and $t_i\{\underline{T} \Rightarrow \Rightarrow nil\} \equiv t_i$. So, we get $FS(P_1, F_1((nil)))=T \neq nil$.

This contradiction proves the Lemma.

Let us consider the functional programming language
$L_2 = (M, C_2, V, \Lambda(C_2, V))$, where $C_2 = M \cup (\Phi \setminus \{cdr\})$.

Lemma 3.2. The function *cdr* is not representable in the language L_2.

Proof. The proof of this Lemma is similar to the proof of Lemma 3.1. Here we consider the work of the interpretation algorithm FS for two cases: $FS(P_2, F_1((T\underline{T})))$ and $FS(P_2, F_1((T\ nil)))$. By induction it can be proved that for any $0 \leq i \leq n$, $\underline{T} \notin C^0(t_i)$, $C^0(t_i)$ does not contain a list containing a sublist with head $\underline{T}$ and $t_i\{\underline{T} \Rightarrow nil\} \equiv t'_i$. So, we get $FS(P_2, F_1((T\ nil)))=(T) \neq (nil)$. Consequently, we get a contradiction, which proves the Lemma.

Definition 3.1. To each $m \in M$ we will correspond a natural number A_m (we will call it the count of atoms of m):

1. If $m=\perp$, then $A_m = 0$;
2. If $m \in Atoms$, then $A_m = 1$;
3. If $m=(m_1 \ldots m_n)$, $m_i \in M$, $i=1,\ldots,n$, $n \geq 0$, then $A_m = A_{m_1} + \ldots + A_{m_n}$.

Let P be a program. Let $\{m_1,\ldots,m_n\}$ be the set of constants of order 0 used in the program P, where $m_i \in M$, $i=1,\ldots,n$, $n \geq 0$. By A_P we will devote the following: $A_P = \max\left\{A_{m_1}, \ldots, A_{m_n}\right\} + 1$.

Let us consider the functional programming language
$L_3 = (M, C_3, V, \Lambda(C_3, V))$, where $C_3 = M \cup (\Phi \setminus \{cons\})$.

Lemma 3.3. The function *cons* is not representable in the language L_3.

Proof. The proof of this Lemma is similar to the proof of Lemma 3.1. Here we consider the work of the interpretation algorithm FS: $FS(P_3, F_1(TT'))$, where $T' = \underbrace{(T \ldots T)}_{A_{P_3}}$. By induction it can be proved that for any $0 \leq i \leq n$ $\max\{A_m \in C^0(t_i)\} \leq A_{P_3}$. So, we get $FS(P_3, F_1(T, T')) \neq \underbrace{(T \ldots T)}_{A_{P_3}+1}$. The contradiction proves the Lemma.

Let us consider the functional programming language
$L_4 = (M, C_4, V, \Lambda(C_4, V))$, where $C_4 = M \cup (\Phi \setminus \{atom\})$.

Lemma 3.4. The function *atom* is not representable in the language L_4.

Proof. This Lemma will be proved by contradiction. Let us assume that the function *atom* is representable in the language L_4. That means there exists a program $P_4 \in \wp(L_4)$, $(F_1 \in V_{[M\to M]})$ such that $f_{P_4}=atom$. We are interested in the result of the interpretation algorithm FS in the following two cases: $FS(P_4, F_1(T))$ and $FS(P_4, F_1((T)))$.

$FS(P_4, F_1(T))$ and $FS(P_4, F_1((T)))$ are well definied, because $atom(T)$ and $atom((T))$ are defined and the interpretation algorithm FS is complete.

If $FS(P_4, F_1(T))\neq T$, then $f_{P_4} \neq atom$, and we will get a contradiction. So, let us assume that $FS(P_4, F_1(T))=T$. We will show that $FS(P_4, F_1((T)))\neq nil$ implying $f_{P_4} \neq atom$ and we will get a contradiction once more. In the term $F_1(T)$, which is an input data of the interpretation algorithm FS, the atom T will be underlined. Namely, we will consider $FS(P_4, F_1(\underline{T}))$ and $FS(P_4, F_1((\underline{T})))$.

We will consider two sequences of terms $t_0, t_1,\ldots$ and $t'_0, t'_1,\ldots$ (in these terms some subterms of order 0 will be double underlined). $t_0\equiv F_1((\underline{T}))$, $t'_0\equiv F_1((T))$ and for any $i \geq 0$, t_{i+1} and t'_{i+1} are correspondingly obtained from t_i and t'_i in the following way:

1. If the leftmost redex r of the term t_i is a subterm of double underlined subterm, then $t'_{i+1}\equiv t'_i$ and the term t_{i+1} is obtained from the term t_i by replacing the redex r with its bundle. In the term t_{i+1} the subterm corresponding to the double underlined subterm, which contains the term r, is double underlined. In the term t_{i+1} all terms, which are double underlined in the term t_i, are double underlined;

2. If the leftmost redex r of the term t_i is not a subterm of double underlined subterm, and if the leftmost redex r' of the term t'_i is a subterm of double underlined subterm, then $t_{i+1}\equiv t_i$ and the term t'_{i+1} is obtained from the term t'_I by replacing the redex r' with its bundle. In the term t'_{i+1} the subterm corresponding to the double underlined subterm, which contains the term r', is double underlined. In the term t'_{i+1} all terms, which are double underlined in the term t'_i, are double underlined;

3. If the leftmost redex r of the term t_i and the leftmost redex r' of the term t'_i are not subterms of double underlined subterms, then the terms t_{i+1} and t'_{i+1} are obtained correspondingly from the terms t_i and t'_i by replacing the redexes r and r' with their bundles. Let r be a β-redex $\lambda x_1 \ldots x_k[\tau_0](\tau_1 \ldots \tau_k)$, where $x_i \in V_{\alpha_i}$, $\tau_i \in \Lambda_{\alpha_i}$, $\tau_0 \in \Lambda$, $\alpha_i \in Types$, $i=1,\ldots,k$, $k \geq 1$. In the bundles of redexes the subterms, which correspond to double underlined subterms of τ_0, are also double underlined. If for any $i=1,\ldots,k$, $k \geq 1$, a subterm of the term τ_i is double underlined, then if in τ_0 a free occurrence of the variable x_i is not in double underlined subterm, then after substitution double underlined subterm of the term t_i is double underlined, otherwise, it is not. Let r be a δ-redex. During the proof of this Lemma the cases of δ-redexes are considered separately and it is denoted, which subterms in bundle of δ-redex, are double underlined. Double underlined subterms of the term t'_{i+1} are obtained similarly;

4. If $t_i \in NF$, $FV(t_i)\cap\{F_1,\ldots,F_n\}\neq\varnothing$ and in the term t_i all free occurrences of the variables $F_1,\ldots,F_n$ stand in double underlined subterms, then

$t_{i+1} \equiv t_i\{\tau_1/F_1,\ldots,\tau_n/F_n\}$ and $t'_{i+1} \equiv t'_i$. In the term t_{i+1} subterms corresponding to double underlined subterms of the term are double underlined;

5. If $t_i \in NF$, $FV(t_i) \cap \{F_1,\ldots,F_n\} \neq \varnothing$, $t'_i \in NF$, $FV(t'_i) \cap \{F_1,\ldots,F_n\} \neq \varnothing$, and in the term t'_i all free occurrences of the variables $F_1,\ldots,F_n$ stand in double underlined subterms, then $t'_{i+1} \equiv t'_i\{\tau_1/F_1,\ldots,\tau_n/F_n\}$ and $t'_{i+1} \equiv t_i$. In the term t'_{i+1} subterms corresponding to double underlined subterms of the term t'_i are double underlined;

6. If $t_i \in NF$, $FV(t_i) \cap \{F_1,\ldots,F_n\} \neq \varnothing$ and if in the term t_i at least one of free occurrences of the variables $F_1,\ldots,F_n$ is not in double underlined subterm, then $t_{i+1} \equiv t_i\{\tau_1/F_1,\ldots,\tau_n/F_n\}$ and $t'_{i+1} \equiv t'_i\{\tau_1/F_1,\ldots,\tau_n/F_n\}$. In the terms t_{i+1} and t'_{i+1} subterms corresponding to double underlined subterms of the terms t_i and t'_i are double underlined.

It is obvious that in the sequences $t_0, t_1, \cdots$ and $t'_0, t'_1, \cdots$ there are no infinite sequences $t_i \equiv t_{i+1} \equiv t_{i+2} \equiv \cdots$ or $t'_i \equiv t'_{i+1} \equiv t'_{i+2} \equiv \cdots$ $(i \geq 0)$. For any $i > 0$ we will double underlin those subterms of order 0, in the term t_i, for which corresponding subterms in the term t'_i are $\perp$, and in the term t'_i we will double underlin those subterms of order 0, for which corresponding subterms in the term t_i are $\perp$. By $\tilde{\tau}$ we will denote the term obtained from the term τ by replacing all double underlined subterms of order 0 with $\perp$.

Then there exists $n>0$ such that t_n=T, because $FS(P_4,F_1(T))=T$ and the term t_{i+1} is either congruent to the term t_i or obtained from t_i by applying one of the steps of the interpretation algorithm FS with input data P_4 and t_i.

By induction it can be proved that for any $0 \leq i \leq n$, $\tilde{t}_i\{\underline{T} \Rightarrow (T) \equiv \tilde{t}'_i$. So, we get $FS(P_4,F_1((T)))=T$, $FS(P_4,F_1((T)))=(T)$ or $FS(P_4,F_1((T)))=\perp$, so, $FS(P_4,F_1((T)))\neq nil$. So, we get a contradiction, which proves the Lemma.

Let us consider the functional programming language $L_5 = (M,\, C_5,\, V,\, \Lambda(C_5, V))$, where $C_5 = M \cup (\Phi \setminus \{if\_then\_else\})$.

Lemma 3.5. The function if_then_else is not representable in the language L_5.

Proof. Let us assume that the function if_then_else is representable in the language L_5. It follows that the function $g \in [M \rightarrow M]$ will be representable in the language L_5 also, where

$$g(m) = \begin{cases} T, & \text{if } m = T, \\ (T), & \text{if } m = nil, \\ \perp, & \text{otherwise.} \end{cases} \qquad m \in M,\ T, nil \in Atoms,$$

We will get a desired contradiction by proving that the function g is not representable in the language L_5. The proof is similar to the proof of Lemma 3.1. Here we consider the action of the interpretation algorithm FS for two cases: $FS(P_5,F_1(\underline{T}))$ and $FS(P_5,F_1(nil))$. By induction it can proved that for any $0 \leq i \leq n$, $\tilde{t}_i\{\underline{T} \Rightarrow nil,\ \underline{nil} \Rightarrow T\} \equiv \tilde{t}'_i$. So, we get $FS(P_5,F_1(nil))=T$, $FS(P_5,F_1(nil))=nil$ or $FS(P_5,F_1(nil))=\perp$ and, so, $FS(P_5,F_1(nil))\neq(T)$. So, we get a contradiction, which proves the Lemma.

Let us consider the functional programming languages

$$L_6 = (M, C_6, V, \Lambda(C_6, V)), \text{ and } L'_6 = (M, C_6, V, \Lambda(C_6, V)),$$

where $C_6 = M \cup (\Phi \setminus \{eq\})$. The language L_6 uses more than two atoms, the language L'_6 uses only two atoms.

Lemma 3.6. The function *eq* is representable in the language L'_6 and is not representable in the language L_6.

The function *eq* is representable in the language L'_6, because it is the least solution of the following equation:

$$F_{eq} = \lambda xy[\mathit{if}\ atom(x)\ then(\mathit{if}\ atom(y)\ then(\mathit{if}\ x\ then\ y\ else(\mathit{if}\ y\ then\ nil\ else\ T))else\ \bot)else\ \bot].$$

Now let us show that the function *eq* is not representable in the language L_6, which uses more than two atoms, $\{a, T, nil\} \subset Atoms$.

It we assume that the function *eq* is representable in the language L_6 , then the function $f \in [M \to M]$ will be representable in the language L_6 also, where

$$f(m) = \begin{cases} T, & \text{if } m = a, \\ a, & \text{if } m = T, \quad m \in M, \quad a, T \in Atoms, \quad a \neq T, nil, \\ \bot, & \text{otherwise.} \end{cases}$$

To get a contradiction, let us prove that the function *f* is not representable in the language L_6. The proof is similar to the proof of Lemma 3.1. Now we consider the work of the interpretation algorithm *FS* for two cases: $FS(P_6,F_1(\underline{a}))$ and $FS(P_6,F_1(T))$. By induction it can be proved that for any $0 \le i \le n$, $t_i\{\underline{a} \Rightarrow T\} \equiv t'_i$. So, we get $FS(P_6, F_1(T)) = T \neq a$, the contradiction proves the Lemma.

Received 21.02.2012

REFERENCES

1. **Martirosyan G.A.** Proceedings of the Conference on Computer Science and Information Technologies (CSIT-2011). Yer., 2011, p. 370–373.
2. **Nigiyan S.A.** // Programming and Computer Software, 1992, v. 17, p. 290–297.
3. **Nigiyan S.A.** // Programming and Computer Software, 1993, v. 19, p. 71–78.
4. **Budaghyan L.E.** Proceedings of the Conference on Computer Science and Information Technologies (CSIT-2005). Yer., 2005, p. 16–19.