Bull. Nov. Comp. Center, Comp. Science, 29 (2009), 107–117
© 2009 NCC Publisher

The language of calculus of computable predicates as a minimal kernel for functional languages

V. I. Shelekhov

Abstract. Logical semantics is a new kind of formal semantics used to describe semantics of pure call-by-value functional languages. For the statement S, the logical semantics $\mathsf{LS(S)}$ is a predicate that is true for the variable values for which the execution of S is finished. Let a specification of the statement S be presented by the Hoare's triple $\{\mathsf{P(x)}\}\,\mathsf{S}\,\{\mathsf{Q(x,y)}\}$. A total correctness of the statement S may be expressed by the formula: $\mathsf{P(x)} \Rightarrow [\,\mathsf{LS(S)(x,y)} \Rightarrow \mathsf{Q(x,y)}\,]\,\&\,\exists \mathsf{y}.\mathsf{LS(S)(x,y)}$. Now, if one has constructed the logical semantics for a functional language, correctness of any functional program supplied with a specification can be proved. In this article, the language of Calculus of Computable Predicates (CCP) is defined as a minimal kernel for constructing the logical semantics of any pure functional language F. The CCP language includes the superposition statement, parallel statement, conditional statement, predicate and array constructor statements. The logical and operational semantics of CCP have been developed. The consistency between these semantics is proved.

Keywords: functional language, Hoare's triple, operational semantics, logical semantics, total correctness of a program.

1. Introduction

Each program includes logic implicitly. Logic is hardly extracted from a program especially for imperative languages. Logical semantics is a new kind of formal semantics that explicitly defines logic of a program.

Let S be a statement in any programming language. The logical formula $\mathsf{L_S}$ denotes a predicate that reflects logic of the statement S. $\mathsf{L_S}$ should be logically equivalent to S. The function $\mathsf{LS} : \mathsf{S} \to \mathsf{L_S}$ defined for each statement S of the language is called *logical semantics* for the language. Logical and operational semantics should be *consistent*: the formula $\mathsf{LS(S)}$ is true for a fixed variable values if and only if the execution of the statement is finished for that values.

Let a program be the statement S, x be input variables (or *arguments*), and y be output variables (or *results*). Our consideration is restricted by the programs with the following sequence of actions: input of arguments, execution of the statement S , and output of results. During the statement S execution, no interactions with the external environment are possible. Such a program implements a function from x to y. Of course, a typical program usually performs inputs and outputs within the execution stage but usually it is possible to restructure the program so that all inputs be before and outputs be after execution. Obviously, it is impossible to restructure in this way a program that implements parallel interacting processes. So, we need to exclude reactive systems from our consideration. Adequate models for reactive systems are the process algebras of R. Milner [2], C.A.R. Hoare [3], etc.

The specification of a program, represented by S with the above restrictions, can be defined by the formula $\mathsf{P(x)} \,\&\, \mathsf{Q(x,y)}$, where $\mathsf{P(x)}$ is called a *precondition* and $\mathsf{Q(x,y)}$ is called a *postcondition.* The precondition should be true before execution of S, and the postcondition – after execution. Thus, a program with a specification may be represented by the well-known Hoare's triple [4]:

$$\{\mathsf{P(x)}\}\,\mathsf{S}\,\{\mathsf{Q(x,y)}\}. \tag{1}$$

Let F be the language with the logical semantics LS. Let Hoare's triple $\{\mathsf{P(x)}\}\,\mathsf{S}\,\{\mathsf{Q(x,y)}\}$ defines a program with a specification where the statement S is written in the language F. Total correctness of the statement S may be expressed by the formula:

$$\mathsf{P(x)} \Rightarrow [\,\mathsf{LS(S)(x,y)} \Rightarrow \mathsf{Q(x,y)}\,] \,\&\, \exists \mathsf{y.LS(S)(x,y)}. \tag{2}$$

The sub-formula $\mathsf{LS(S)(x,y)} \Rightarrow \mathsf{Q(x,y)}$ states that the results y of program execution satisfy the postcondition. This sub-formula expresses the main law of consistency between a program and a specification: the program should satisfy its specification. The sub-formula $\exists \mathsf{y.LS(S)(x,y)}$ states that the program execution is fulfilled for the arguments x.

Now, if the logical semantics for some language has been constructed, one can prove correctness of any program supplied with a specification using formula (2). It is possible to construct logical semantics for a pure call-by-value functional language. The development of logical semantics for any existing functional languages is a difficult task. So it is better to start from a simple subset of the language. Our aim is to define a universal subset for all functional languages. The predicate calculus[1] may be used as a universal basis for functional languages. In this article, the language of Calculus of Computable Predicates (CCP) is introduced as a minimal kernel for constructing the logical semantics of any pure functional language F. A construct of the language F may be defined as a hierarchical denotation through the constructs of the CCP language. So, there exists an extending language chain: $\mathsf{CCP} \subset \mathsf{F_1} \subset \mathsf{F_2} \subset \ldots \subset \mathsf{F}$ where each $\mathsf{F_j}$ is defined through the constructs of the previous language in the chain.

The rest of this article is organized as follows. Section 2 presents the related works. The restrictions on the functional languages are formulated.

[1]Here and further we assume the predicate calculus of high orders.

Section 3 defines the CCP language, its logical and operational semantics. It includes the superposition, parallel and conditional statements, the predicate and array constructor statements. Expressions and constants are not allowed. Section 4 concludes the article with the remarks on defining the semantics for an extending language chain.

2. Related work

Formal semantics of a programming language can be considered as an elegant and powerful instrument to prove the properties of a program but not for an existent imperative language. Its denotational semantics [7] is highly complicated, and axiomatic semantics [4, 8] may be constructed only for a simple subset of the language. This point of view, claimed by John Backus in his Turing's lecture [1] 30 yeas ago, still remains valid today.

Logical semantics can be evidently defined for logic programming languages. This is a usual mathematical semantics used in logical deduction of a logical program. So there is no need to define logical semantics here. The construction of logical semantics may be useful only for non-logical features of logic programming. For example, the semantics of depth-first logic programming with version negation as failure was described in fourth-valued logic [5]. Logical semantics can easily be defined for the language of predicative programming by Eric Hehner [11].

The development of logical semantics for an exploitable imperative language is an extremely difficult problem. There are some restrictions on a functional language in order the construction of the logical semantics be possible. First, the functional language should be a pure call-by-value language. Lazy evaluations [13] are not allowed. Second, each function in a program should be total. So the language should be strictly typed like the PVS specification language [14]. Otherwise only three-valued logic (true, false, undefined) may be used to define logical semantics for that language.

The language of predicate calculus is considered as a programming language in R.Kowalski's article [9] acknowledged as the basic for logic programming. The CCP language is defined as a computable part of the predicate calculus but for functional languages. The concept of semantic programming [10] is also based on interpretation of the predicate calculus as a programming language; a program is executed there in the style of constraint programming [12].

3. The language of calculus of computable predicates. Operational and logical semantics

We are trying to define the CCP language as the most simple and universal language capable to represent any algorithm. Our aim is to construct the logical semantics for this language and investigate its properties.

3.1. Types

The type collection in the CCP language includes primitive types, a subset of some type, and structure types. *Primitive types* are BOOL, INT, REAL, and CHAR. A type S which is a *subset of the type* T is defined by:

$$\mathsf{S} = \mathsf{SUBSET}(\mathsf{T}, \mathsf{x}, \mathsf{P}, \mathsf{d}). \tag{3}$$

Here x is a variable of type T, P is a predicate name, and d is a possibly empty list of variables. The type S is defined as $\mathsf{S} = \{\mathsf{x} \in \mathsf{T} \mid \mathsf{P}(\mathsf{x}, \mathsf{d})\}$. The predicate $\mathsf{P}(\mathsf{x}, \mathsf{d})$ should belong to the CCP language. The variables of the list d are called parameters of the type S. For example, the type $\mathsf{DIAP}(\mathsf{n}) = \mathsf{SUBSET}(\mathsf{INT}, \mathsf{x}, \mathsf{INT1\_n}, \mathsf{n})$ defines the interval of natural numbers from 1 to n, where $\mathsf{INT1\_n}(\mathsf{x}, \mathsf{n}) \;\cong\; \mathsf{x} \geq 1 \,\&\, \mathsf{x} \leq \mathsf{n}$.

Let X, Y, and Z be type names, D and E be lists of type names. The *structure types* are the following:

$$\begin{array}{ll} \textit{product type} & \mathsf{Z} = \mathsf{X} \times \mathsf{Y} = \{(\mathsf{x}, \mathsf{y}) \mid \mathsf{x} \in \mathsf{X}, \mathsf{y} \in \mathsf{Y}\}, \qquad (4) \\ \textit{union type} & \mathsf{Z} = \mathsf{X} + \mathsf{Y} = \{(1, \mathsf{x}) \mid \mathsf{x} \in \mathsf{X}\} \cup \{(2, \mathsf{y}) | \mathsf{y} \in \mathsf{Y}\}, \qquad (5) \\ \textit{set type} & \mathsf{Z} = \mathsf{SET}(\mathsf{X}) = \{\mathsf{z} \mid \mathsf{z} \subseteq \mathsf{X}\}, \qquad (6) \\ \textit{predicate type} & \mathsf{Z} = \mathsf{PRED}(\mathsf{D}{:}\mathsf{E}) = \{\varphi(\mathsf{d}{:}\ \mathsf{e}) \mid \mathsf{d} \in \mathsf{D}, \mathsf{e} \in \mathsf{E}, \varphi \in \mathsf{CCP}\}. \ (7) \end{array}$$

In the definition of a predicate type, $\varphi(\mathsf{d}{:}\ \mathsf{e})$ denotes a predicate $\varphi(\mathsf{d}, \mathsf{e})$, where d and e are lists of variables. If the types D are finite and all the predicates $\varphi(\mathsf{d}{:}\ \mathsf{e})$ are total and single-valued, that is $\forall \mathsf{d} \in \mathsf{D}\ \exists ! \mathsf{e}.\ \varphi(\mathsf{d}, \mathsf{e})$, then $\varphi(\mathsf{d}{:}\ \mathsf{e})$ may be interpreted as an *array* with *indexes* d and *array element* e.

The sequence and tree types may be defined recursively. Pointers are not allowed.

3.2. Operational and logical semantics

Operational semantics of the CCP language is described in the form of a metaprogram defined in a metalanguage. The metaprogram memory is a collection of memory sections. Each *memory section* consists of variables (arguments, results, and locals) for the definition of some predicate. Let s be a memory section, a be a variable name, and v be a value. The construct $\mathsf{s}[\mathsf{a}]$ denotes the value of the variable a in the section s. The assignment statement $\mathsf{s}[\mathsf{a}] := \mathsf{v}$ assigns the value v to the variable a in the section s. In the metalanguage, there is the multi-assignment statement $\mathsf{s}[\mathsf{x}] := \mathsf{w}$, where x is the list of variable names and w is the list of values. The statement $\mathsf{s} = \mathsf{newSect}(\mathsf{A})$ creates a new section s in the memory for the predicate A.

For any CCP construct $\mathsf{H}(\mathsf{x},\mathsf{y})$, $\mathsf{RUN}(\mathsf{H},\mathsf{x},\mathsf{y})$ denotes the following statement: for fixed values of x and y, there exists an execution $\mathsf{H}(\mathsf{x},\mathsf{y})$ which is fulfilled and the results of execution are the values of variables y. According to this definition, the non single-valued execution of $\mathsf{H}(\mathsf{x},\mathsf{y})$ is possible. The *consistency* property $\mathsf{Cons}(\mathsf{H})$ of logical and operational semantics is defined as follows:

$$\mathsf{Cons}(\mathsf{H}) \;\cong\; \forall \mathsf{x}\, \forall \mathsf{y}\; (\mathsf{LS}(\mathsf{H}(\mathsf{x},\mathsf{y})) \;\Leftrightarrow\; \exists \mathsf{y}'\; (\mathsf{RUN}(\mathsf{H},\mathsf{x},\mathsf{y}') \;\&\; \mathsf{eq}(\mathsf{y},\mathsf{y}'))). \tag{8}$$

The assertion $\mathsf{eq}(\mathsf{y},\mathsf{y}')$ is the equality $\mathsf{y} = \mathsf{y}'$ for variables of all types except the predicate type. For the result φ of the predicate type, $\mathsf{eq}(\varphi,\varphi')$ denotes identity of predicates: $\forall \mathsf{d} \forall \mathsf{e}.\, \varphi(\mathsf{d},\mathsf{e}) \equiv\; \varphi'(\mathsf{d},\mathsf{e})$.

3.3. Predicate call. Definition of a predicate

A *predicate call* is a statement of the form $\mathsf{A}(\mathsf{z}\colon \mathsf{u})$, where A is the predicate name or the name of a variable of the predicate type, z and u are the variable lists. The list z may be empty. The variables in the list u cannot belong to z. We use the terms *call arguments* and *call results* to denote the variables of the lists z and u, respectively. The logical semantics of the call $\mathsf{A}(\mathsf{z}\colon \mathsf{u})$ is simply defined as:

$$\mathsf{LS}(\mathsf{A}(\mathsf{z}\colon \mathsf{u})) \;\cong\; \mathsf{A}(\mathsf{z},\mathsf{u}). \tag{9}$$

The operational semantics of the call $\mathsf{A}(\mathsf{z}\colon \mathsf{u})$ is defined by the procedure call $\mathsf{runCall}(\mathsf{s},\mathsf{A}(\mathsf{z}\colon \mathsf{u}))$ in the metalanguage.

A program in the CCP language is a collection of predicate definitions. A predicate definition is a construct of the form:

$$\mathsf{A}(\mathsf{x}\colon \mathsf{y}) \;\equiv\; \mathsf{K}(\mathsf{x}\colon \mathsf{y}), \tag{10}$$

where A is the name of the predicate defined by this construct, x and y are the variable lists, $\mathsf{K}(\mathsf{x}\colon \mathsf{y})$ denotes a superposition, parallel or conditional statement. The variables in the x and y lists are different. They are called *predicate arguments* and *predicate results*, respectively.

Let the predicate call $\mathsf{A}(\mathsf{z}\colon \mathsf{u})$ be executed in the memory section q which includes the variables of the lists z and u. Let the predicate A be defined by (10). The execution of the call $\mathsf{A}(\mathsf{z}\colon \mathsf{u})$ performed by $\mathsf{runCall}(\mathsf{s},\mathsf{A}(\mathsf{z}\colon \mathsf{u}))$ is defined by the following sequence of statements:

$$\begin{aligned} &\mathsf{s} = \mathsf{newSect}(\mathsf{A}); \\ &\mathsf{s}[\mathsf{x}] := \mathsf{q}[\mathsf{z}]; \\ &\mathsf{runStat}(\mathsf{s},\; \mathsf{K}(\mathsf{x}\colon \mathsf{y})); \\ &\mathsf{q}[\mathsf{u}] := \mathsf{s}[\mathsf{y}] \end{aligned} \tag{11}$$

where the execution of the statement $\mathsf{K}(\mathsf{x}\colon \mathsf{y})$ is expressed by the procedure call $\mathsf{runCall}(\mathsf{s}, \mathsf{K}(\mathsf{x}\colon \mathsf{y}))$ in the metalanguage. At the end of the execution of (11) the logical formula $\mathsf{x} = \mathsf{z}$ & $\mathsf{u} = \mathsf{y}$ is true.

Lemma 1. For the predicate definition (10) and any procedure call $\mathsf{A}(\mathsf{z}\colon \mathsf{u})$, $\mathsf{Cons}(\mathsf{K}) \Rightarrow \mathsf{Cons}(\mathsf{A}(\mathsf{z}\colon \mathsf{u}))$.

3.4. Superposition statement

A *superposition statement* is the construct:

$$\mathsf{B}(\mathsf{x}\colon \mathsf{z});\ \mathsf{C}(\mathsf{z}\colon \mathsf{y}) \tag{12}$$

Here x, y, and z are variable lists, $\mathsf{B}(\mathsf{x}\colon \mathsf{z})$ and $\mathsf{C}(\mathsf{z}\colon \mathsf{y})$ are predicate calls. The variables in the lists z and y are different; they cannot belong to the list x which may be empty. If B or C is the name of a variable of the predicate type, then it belongs to the list x. The logical semantics is defined as follows:

$$\mathsf{LS}(\mathsf{B}(\mathsf{x}\colon \mathsf{z});\ \mathsf{C}(\mathsf{z}\colon \mathsf{y})) \ \cong\ \exists \mathsf{z}.\ (\mathsf{B}(\mathsf{x},\ \mathsf{z})\ \&\ \mathsf{C}(\mathsf{z},\ \mathsf{y})). \tag{13}$$

Let the statement $\mathsf{B}(\mathsf{x}\colon \mathsf{z});\ \mathsf{C}(\mathsf{z}\colon \mathsf{y})$ be executed in the memory section s which includes the variables of the lists x, y, and z. The execution of the statement performed by $\mathsf{runStat}(\mathsf{s}, \mathsf{B}(\mathsf{x}\colon \mathsf{z});\ \mathsf{C}(\mathsf{z}\colon \mathsf{y}))$ is defined by the following statements in the metalanguage:

$$\begin{aligned}&\mathsf{runCall}(\mathsf{s}, \mathsf{B}(\mathsf{x}\colon \mathsf{z}));\\ &\mathsf{runCall}(\mathsf{s}, \mathsf{C}(\mathsf{z}\colon \mathsf{y}))\end{aligned} \tag{14}$$

Lemma 2. For the superposition statement (12),
$\mathsf{Cons}(\mathsf{B})\ \&\ \mathsf{Cons}(\mathsf{C}) \Rightarrow \mathsf{Cons}(\mathsf{H})$, where H is $\mathsf{B}(\mathsf{x}\colon \mathsf{z});\ \mathsf{C}(\mathsf{z}\colon \mathsf{y})$.

3.5. Parallel statement

A *parallel statement* is the construct:

$$\mathsf{B}(\mathsf{x}\colon \mathsf{y}) \,||\, \mathsf{C}(\mathsf{x}\colon \mathsf{z}) \tag{15}$$

where x, y, and z are variable lists, $\mathsf{B}(\mathsf{x}\colon \mathsf{y})$ and $\mathsf{C}(\mathsf{x}\colon \mathsf{z})$ are predicate calls. The variables in the lists y, and z are different; they cannot belong to the list x which may be empty. If B or C is the name of a variable of the predicate type, then it belongs to the list x. The logical semantics is defined as follows:

$$\mathsf{LS}(\mathsf{B}(\mathsf{x}\colon \mathsf{y}) \,||\, \mathsf{C}(\mathsf{x}\colon \mathsf{z})) \ \cong\ \mathsf{B}(\mathsf{x}, \mathsf{y})\ \&\ \mathsf{C}(\mathsf{x}, \mathsf{z}).$$

Let the statement $\mathsf{B}(\mathsf{x}\colon \mathsf{y}) \,||\, \mathsf{C}(\mathsf{x}\colon \mathsf{z})$ be executed in the memory section s which includes the variables of the lists x, y, and z. Execution of the statement performed by $\mathsf{runStat}(\mathsf{s}, \mathsf{B}(\mathsf{x}\colon \mathsf{y}) \,||\, \mathsf{C}(\mathsf{x}\colon \mathsf{z}))$ is defined by the following statement:

$$\mathsf{runCall(s,\ B(x\colon y))}\,|| \tag{16}$$
$$\mathsf{runCall(s,\ C(x\colon z))} \tag{17}$$

The parallel composition || defines independent execution of two parts without interaction between them. So an effect of the parallel composition is the conjunction of the effects of the parts.
Lemma 3. For the parallel statement (15), $\mathsf{Cons(B)}\,\&\,\mathsf{Cons(C)} \Rightarrow \mathsf{Cons(H)}$, where H is $\mathsf{B(x\colon y)}\,||\,\mathsf{C(x\colon z)}$.

3.6. Conditional statement

A *conditional statement* is the construct:

$$\textbf{if}\ \mathsf{(b)\ B(x\colon y)}\ \textbf{else}\ \mathsf{C(x\colon y)} \tag{18}$$

where x and y are variable lists, $\mathsf{B(x\colon y)}$ and $\mathsf{C(x\colon y)}$ are predicate calls, the variable b is of BOOL type; b does not belong to the lists x and y. The variables in the list y are different; they cannot belong to the list x. which may be empty. If B or C is the name of a variable of the predicate type, then it belongs to the list x. The logical semantics is defined as follows:

$$\mathsf{LS}(\textbf{if}\ \mathsf{(b)\ B(x\colon y)}\ \textbf{else}\ \mathsf{C(x\colon y)}) \ \cong\ (\mathsf{b} \Rightarrow \mathsf{B(x,\ y)})\ \&\ (\neg\mathsf{b} \Rightarrow \mathsf{C(x,\ y)}) \tag{19}$$

Let the statement **if** $\mathsf{(b)\ B(x\colon y)}$ **else** $\mathsf{C(x\colon y)}$ be executed in the memory section s which includes the variable b and the variables of the lists x and y. The execution of the statement performed by $\mathsf{runStat(s,}\textbf{if}\ \mathsf{(b)\ B(x\colon y)}\ \textbf{else}\ \mathsf{C(x\colon y))}$ is defined by the following statement in the metalanguage:

$$\textbf{if}\ \mathsf{(s[b])\ runCall(s, B(x\colon y))}\ \textbf{else}\ \mathsf{runCall(s, C(x\colon y))} \tag{20}$$

Lemma 4. For the conditional statement (18),
$\mathsf{Cons(B)}\,\&\,\mathsf{Cons(C)} \Rightarrow \mathsf{Cons(H)}$, where H is **if** $\mathsf{(b)\ B(x\colon y)}$ **else** $\mathsf{C(x\colon y)}$.

3.7. Predicate constructor

A *predicate constructor* statement is the basic predicate:

$$\mathsf{Pred(x, B\colon A)}, \tag{21}$$

where x is a possibly empty variable list, B is a predicate name, A is the name of a variable of the predicate type $\mathsf{PRED(Y\colon Z)}$; Y and Z are lists of types. For the predicate $\mathsf{B(x, y\colon z)}$, y and z are variable lists of types Y and Z, respectively; the list y may be empty. The basic predicates Pred and Array are not allowed instead of B in (21). The logical semantics is defined as follows:

$$\mathsf{LS(Pred(x, B: A))} \quad \cong \quad \forall \mathsf{y} \forall \mathsf{z}.\ (\mathsf{A(y,z)} \equiv \mathsf{B(x,y,z)}). \tag{22}$$

Let the predicate call $\mathsf{Pred(x, B: A)}$ be executed in the memory section s which includes the variable A and the variables of the list x. The execution of the statement performed by $\mathsf{runCall(s, Pred(x, B: A))}$ is defined by the following statement:

$$\mathsf{s[A]} \ := \ \mathsf{newDef(s[x], B)} \tag{23}$$

The result of the function newDef is a new predicate name $\mathsf{A_x}$ which is unique in the process of program execution. The definition, constructed by newDef, of a new predicate with the name $\mathsf{A_x}$ is the following:

$$\mathsf{A_x(y: z)} \ \equiv \ \mathsf{equ(x^{\sim}: t);\ B(t, y: z)}, \tag{24}$$

where $\mathsf{x^{\sim}} = \mathsf{s[x]}$, the basic predicate $\mathsf{equ(x^{\sim}: t)}$ performs the assignment $\mathsf{s[t]} := \mathsf{x^{\sim}}$.

Lemma 5. $\mathsf{Cons(Pred)}$.

Lemma 6. For the predicate constructor (21), $\mathsf{Cons(B)} \Rightarrow \mathsf{Cons(A)}$.

3.8. Array constructor

An *array constructor* statement is the basic predicate:

$$\mathsf{Array(x, B: A)}, \tag{25}$$

where x is a possibly empty variable list, B is a predicate name different from Pred and Array, A is the name of a variable of the predicate type $\mathsf{PRED(Y{:}Z)}$; Y and Z are lists of types. For the predicate $\mathsf{B(x, y: z)}$, y and z are variable lists of types Y and Z, respectively. The types of the list Y are finite. The logical semantics is defined as follows:

$$\mathsf{LS(Array(x, B: A))} \quad \cong \quad \forall \mathsf{y} \forall \mathsf{z}.\ (\mathsf{A(y,z)} \equiv \mathsf{B(x,y,z)})). \tag{26}$$

Note that the logical semantics of Array and Pred is the same.

Let the predicate call $\mathsf{Array(x, B: A)}$ be executed in the memory section s which includes the variable A and the variables of the list x. The execution of the statement performed by $\mathsf{runCall(s, Array(x, B: A))}$ is defined by the following statements:

$$\begin{array}{l} \mathsf{s[A] = newArray(Y, Z);} \\ \textbf{forAll}\ \mathsf{y \in Y}\ \textbf{do}\ \mathsf{runCall(s, B(x, y: s[A][y]))}\ \textbf{end} \end{array} \tag{27}$$

The function $\mathsf{newArray(Y, Z)}$ creates a new array $\mathsf{A^{\sim}}$ in the memory. There is the following relation between the predicate $\mathsf{A(y: z)}$ and the array $\mathsf{A^{\sim}}$:

$$\mathsf{A}(\mathsf{y}\colon \mathsf{z}) \equiv \mathsf{A}^{\sim}[\mathsf{y}] = \mathsf{z}. \tag{28}$$

The statement **forAll** iterates all indexes y from the type list Y. The statement between **do** and **end** is executed independently, possibly in parallel, for each index list $\mathsf{y} \in \mathsf{Y}$. If the effect of the statement $\mathsf{S}(\mathsf{y}\colon \mathsf{z})$ is defined by the formula $\mathsf{F}(\mathsf{y})$, then the effect of the statement **forAll** $\mathsf{y} \in \mathsf{Y}$ **do** $\mathsf{S}(\mathsf{y}\colon \mathsf{z})$ **end** is defined by the formula $\forall \mathsf{y} \in \mathsf{Y}.\, \mathsf{F}(\mathsf{y})$.
Lemma 7. For the array constructor (25), $\mathsf{Cons}(\mathsf{A})$.
Lemma 8. Let the predicate $\mathsf{B}(\mathsf{x}, \mathsf{y}\colon \mathsf{z})$ be total and single-valued: $\forall \mathsf{x}\ \forall \mathsf{y} \in \mathsf{Y}\ \exists! \mathsf{z}\ \mathsf{B}(\mathsf{x}, \mathsf{y}\colon \mathsf{z})$. For the array constructor (25),
$\mathsf{Cons}(\mathsf{B}) \Rightarrow \mathsf{Cons}(\mathsf{Array})$.

3.9. Program

A program in the CCP language is a collection of predicate definitions. For any procedure call used in the program, the procedure name should be either defined by a predicate definition belonging to the program, or a formal parameter, or a name of a basic predicate.

A *basic predicate* is used to denote a standard operator on the values of types defined in the CCP language. For the primitive types, two kinds of basic predicates for the equality operator are defined: $= (\mathsf{x}\colon \mathsf{y})$ and $= (\mathsf{x}, \mathsf{y}\colon \mathsf{b})$, where b is of BOOL type, x and y are of a primitive type. A basic predicate with an empty argument list denotes a constant of a primitive type. The predicate and array constructors are also basic predicates.

A collection of the predicate definitions of a program may be recursive. Let B and C be defined predicates. If there is a call of the predicate C in the definition of the predicate B, we use the relation $\mathsf{depend}(\mathsf{B}, \mathsf{C})$ to denote the *immediate dependence* B on C. The predicate B is *defined through* the predicate C, $\mathsf{def}(\mathsf{B}, \mathsf{C})$, if there exists a chain of defined predicates $\mathsf{D}_1, \dots, \mathsf{D}_\mathsf{n} (\mathsf{n} > 0)$, where $\mathsf{B} = \mathsf{D}_1, \mathsf{C} = \mathsf{D}_\mathsf{n}$, and the relation $\mathsf{depend}(\mathsf{D}_\mathsf{j}, \mathsf{D}_{\mathsf{j}+1})$ is true for $\mathsf{j} = 1, \dots, \mathsf{n} - 1$.

The predicate B is *recursive* if the relation $\mathsf{def}(\mathsf{B}, \mathsf{B})$ is true. For a recursively defined predicate B, a *recursive ring* of predicates is the predicate set $\mathsf{rec}(\mathsf{B}) = \{\mathsf{C} \mid \mathsf{def}(\mathsf{B}, \mathsf{C})\, \&\, \mathsf{def}(\mathsf{C}, \mathsf{B})\}$. Obviously, if $\mathsf{C} \in \mathsf{rec}(\mathsf{B})$ then $\mathsf{rec}(\mathsf{B}) = \mathsf{rec}(\mathsf{C})$. For the statements $\mathsf{B}(\mathsf{x}\colon \mathsf{z}); \mathsf{C}(\mathsf{z}\colon \mathsf{y})$, $\mathsf{B}(\mathsf{x}\colon \mathsf{y}) || \mathsf{C}(\mathsf{x}\colon \mathsf{z})$, and **if** (b) $\mathsf{B}(\mathsf{x}\colon \mathsf{y})$ **else** $\mathsf{C}(\mathsf{x}\colon \mathsf{y})$, recursion is allowed only through the defined predicates B and C. Recursion through the condition b is prohibited. When B (or C) is a variable of the predicate type, recursion through B (or C) is prohibited.

Let a recursive ring be represented by the definitions:

$$\mathsf{A}_\mathsf{i}(\mathsf{x}_\mathsf{i}\colon \mathsf{y}_\mathsf{i}) \equiv \mathsf{K}_\mathsf{i}(\mathsf{x}_\mathsf{i}\colon \mathsf{y}_\mathsf{i});\quad \mathsf{i} = 1, \dots, \mathsf{n};\ \mathsf{n} > 0. \tag{29}$$

Here, all variables in the lists x_i and $\mathsf{y}_\mathsf{i} (\mathsf{i} = 1, \dots, \mathsf{n};\ \mathsf{n} > 0)$ are different. A *graphic* of the predicate $\mathsf{B}(\mathsf{x}\colon \mathsf{y})$ is $\mathsf{Gr}(\mathsf{B}) = \{(\mathsf{x}, \mathsf{y}) \mid \mathsf{LS}(\mathsf{B}(\mathsf{x}\colon \mathsf{y}))\}$. Let

$G_j = Gr(A_j), V_j = Gr(K_j), i = 1, \dots, n$,
$G = (G_1, G_2, \dots, G_n)$, and
$V = (V_1, V_2, \dots, V_n)$.
The system (29) of predicate definitions is equivalent to the equation for vector-graphics of predicates:

$$G = V(G). \tag{30}$$

The set of of vector-graphics with the relation "$\subseteq$" on vector-graphics and the empty vector-graphic $(\emptyset, \emptyset, \dots, \emptyset)$ is the complete lattice. It is easy to prove that the graphics for superposition, parallel, and conditional statements are continuous functions with respect to graphics for the predicates B and C. Let $G^0 = \emptyset, G^{m+1} = V(G^m), m \geq 0$. In accordance with the Kleene fixed point theorem [6], the decision of equation (30) is the least fixed point of the function V, $G = lfp(V)$, that is equal to the supremum of the ascending chain $\{G^m\}_{m \geq 0}$.

Let the recursive predicate $D(z\colon u)$ belong to the recursive ring (29) that is $D = A_j$ for some j. The logical semantics of the predicate D is defined as follows:

$$LS(D(z\colon u)) \cong (z, u) \in pr(j, lfp(V)), \tag{31}$$

where $pr(j, G) = G_j$.
Lemma 9. Let the calls of the predicates $E_1, E_2, \dots, E_s$ be used in the predicate definitions of the recursive chain (29), the names $E_1, E_2, \dots, E_s$ be different from $A_1, A_2, \dots, A_n$. Then
$\forall k = 1, \dots, s. Cons(E_k) \Rightarrow \forall j = 1, \dots, n. Cons(A_j)$.
Theorem 1. For any predicate call $Array(x, B\colon A)$ in a program, let the predicate $B(x, y\colon z)$ be total and single-valued: $\forall x\ \forall y \in Y\ \exists! z\ B(x, y, z)$. Let the property $Cons(\varphi)$ be true for any basic predicate φ called in a program. Let a program be executed by the predicate call $D(u\colon v)$. Then $Cons(D)$ is true.

Full proofs for this theorem and above lemmas can be found in [15].

4. Conclusion

The development of logical semantics for any existing functional language is a difficult problem. The CCP language is defined in this article as a minimal kernel for constructing the logical and operational semantics of any pure call-by-value functional language F. Each construct of the F language can be defined as a hierarchical denotation trough the constructs of the CCP language. It is possible to construct an extending language chain: $CCP \subset F_1 \subset F_2 \subset \dots \subset F$, where each F_j is defined through the constructs of the previous language in the chain. Let a new construct C be defined through some composition Q in the previous language F_{j-1}. Logical semantics $LS(C)$

can be obtained by equivalent transformation of the formula $\mathsf{LS}(\mathsf{Q})$. To obtain operational semantics for the new construct C, one can write a program in the metalanguage for the composition Q and make equivalent transformation of the program. According to definition (8), the consistency property of logical and operational semantics for the construct C remains true after any equivalent transformation.

References

[1] Backus J. Can programming be liberated from the von Neumann style? A Functional Style and Its Algebra of Programs // Communs. of the ACM. — 1978. — Vol. 21, N 8. — P. 613–641.

[2] Milner R. A Calculus of Communicating Systems // Lect. Notes Comput. Sci. — 1980. — Vol. 92.

[3] Hoare C.A.R. Communicating Sequential Processes. — 1985.

[4] Hoare C. A. R. An axiomatic basis for computer programming // Communs. of the ACM. — 1969. — Vol. 12, N 10. — P.576–585.

[5] Andrews J. H. A logical semantics for depth-first Prolog with ground negation // Theor. Comput. Sci. — 1997. — Vol. 184. — P. 105–143.

[6] Kleene S. C. Introduction to Metamathematics. — New York, 1952.

[7] Scott D.S., Strachey C. Towards a mathematical semantics for computer languages // Computers and Automata. — 1971. — P. 19–46.

[8] Floyd R.W. Assigning meanings to programs // Proceedings Symposium in Applied Mathematics, Mathematical Aspects of Computer Science / Ed. by J.T. Schwartz. — AMS, 1967. — P. 19–32.

[9] Kowalski R. Predicate Logic as Programming Language // IFIP Congress, Stockholm, North Holland Publishing Co. — 1974. — P. 569–574.

[10] Goncharov S.S., Ershov Yu.L., Sviridenko D.I. Semantic programming // Proc. of 10 IFIP 86. — 1986. — P. 1093–1100.

[11] E.C.R.Hehner. A Practical Theory of Programming, second edition. — 2004.— http://www.cs.toronto.edu/∼hehner/aPToP/

[12] Mantsivoda A. Flang: A Functional-Logic Language // Lect. Notes Comput. Sci. — 1992.— Vol. 567. — P.257–270.

[13] Launchbury J. A Natural Semantics for Lazy Evaluation. // POPL 93. — 1993. — P. 144–154.

[14] PVS Language Reference. Version 2.4 // SRI International. — 2001. —- http://pvs.csl.sri.com/doc/pvs-language-reference.pdf.

[15] Shelekhov V. Calculus of Computable Predicates. — Novosibirsk, 2007. — 24p. — (Prepr. / IIS SB RAS; N 143). (In Russian).