FORMAL PROGRAMMING MACHINES OF MODERN PROGRAMMING LANGUAGES

Let's consider the use of algebraic interpretation of language constructs for PL C, Lisp and Prolog as the most well-known algorithmic languages, which are representatives of languages of the universal, functional and logical type, respectively.

Study of C language constructs

Control constructs of the C++ (C) language are represented by the following algebraic system:

$$FC{+}{+} = \langle AC{+}{+}, \Omega_{C++}, RC{+}{+} \rangle,$$

in which AC++ = { f1, f2, …, fi, … , fn, … } is a set of language constructs described by the generative grammar Gc++ of the C++ language and describing the control of the program structure.

Let's consider an example of an optimizing transformation of a program fragment in C++ (C) using the following conventions. We denote the sequential composition of C language operations, interpreted as the sequential execution of operators, as follows:

$$x_1;\ x_2;\ \ldots\ x_n;\ = f^{+}(x_1, x_{2,} \ldots, x_n).$$

Due to the associativity of this operation and in order to simplify the writing of expressions, we will omit this operation (using the syntax of the left side of the above expression).

The designations required for transformations are summarized in the following table (Table 1).

Table 1

Notation of C-constructs in the form of program terms

N	***Designation of C-design in a programme***	***Designation in the form software term***
1	unsigned x1, ..., xn ;	f0 (x1, ..., xn)
2	if (x1) x2 else x3 ;	f1 (x1, x2, x3)
3	if (x1) x2 ;	f2 (x1, x2)
4	{ x1; ...; xn}	f3 (x1, ..., xn)
5	x1++ ;	f4(x1)
6	x1-- ;	f5(x1)
7	x1 - x2 ;	f6 (x1, x2)

8	! x1 ;	f7 (x1)
9	x1 && x2	f8 (x1, x2)
10	x1 += x2 ;	f9 (x1, x2)

Let the set of operationsΩ^{FC++}contains the following elements:

$\omega_0(x_1, x_2) = x_1; x_2;$ - sequential composition operation,

$\omega_1^j(x_0, f_i(x_1, \ldots, x_j, \ldots x_n)) = f_i(x_1, \ldots, x_0, \ldots, x_n);$ - substitution operation.

The RFC++ relationship set will use the following binary relationships:

= - equivalence,$\neq$- non-equivalence,$\perp$- independence.

Algebra of memory statesΩ_{c++}will be represented by the following system of sets:

$$Q_{C++} =< A^Q_{C++}, \Omega^Q_{C++}, R^Q_{C++} >,$$

Where $A^Q_{C++} = \{Line\_1, L\_Flag, X\}$, $\Omega^Q_{C++} = \{+\}$ - composition, $R^Q_{C++} = \{=, \neq, \perp\}$ - relations of equivalence, non-equivalence and independence, respectively.

Let's look at a specific example. Let it be necessary to prove the equivalence of the two C-programs presented below.

```
Program 1-1 Program 1-2
unsigned Line_1, L_Flag, X; unsigned Line_1, L_Flag,X;
// ... //...
Line_1++; Line_1 += X;
if ( L_Flag ) Line_1 += X
else
if ( !L_Flag && Line_1 ) {
Line_1 += X;
L_Flag = 0;
};
Line_1--;
```

For the proof, we will use the following axioms and theorems of applied calculus, interpreted into the proposed system of algebras.

(1-1) Initialization axiom:

$$\forall x_i: \quad f_0(x_i, \ldots, x_n) = f_0(x_i, \ldots, x_n); f_6(x_i, 0); \ldots f_6(x_n, 0);$$

(1-2) Theorem on the permutation of independent syntactic structures (proved by complete induction on memory elements for various operations):

$$\forall f_i, f_j \quad f_i(x_1, \ldots, x_n) \perp f_j(x_t, \ldots, x_m) \quad =>$$

$$[\, f_i(x_1, \ldots, x_n); f_j(x_t, \ldots, x_m) = f_j(x_t, \ldots, x_m); f_i(x_1, \ldots, x_n);\,].$$

(1-3') Axiom about the conditional operator:

$$f_6(x_i, 0); f_1(x_i, x_j, x_k) = f_6(x_i, 0); x_k \; .$$

(1-4') Axiom about the abbreviation of the conditional operator:

$$f_6(x_i, 0); f_2(x_i, x_k) = x_k \; .$$

(1-5') Substitution axiom:

$$f_6(x_i,0); f_i(x_i,\ldots,x_n) \;=\; f_6(x_i,0); f_i(0,\ldots,x_n) \quad .$$

(1-6') Axioms of elementary mathematical logic:

$$f_7(0)=1; \quad f_7(1)=0;\; f_8(1,x_i)=x_i;\; f_8(0,x_i)=0; f_8(x_i,x_j)=f_8(x_j,x_i) \;.$$

(1-7') Reassignment Axiom:

$$f_6(x_1,x_n); f_6(x_1,x_m)=f_6(x_1,x_m) \quad .$$

(1-8') Theorem on arithmetic expressions (proven in the axiomatics of formal arithmetic):

$$f_4(x_i); f_9(x_i,x_j); f_5(x_i); \;=\; f_9(x_i,x_j) \quad .$$

In the listed expressions, the axioms marked with numbers with a prime refer to optimizing axioms that reduce the text of the program and, accordingly, its running time.

To prove the equivalence of the programs, we write Program 1-1 in algebraic notation.

Step 1 (initial program structure)

```
$f_0(x_1,x_2,x_3);$                       // unsigned Line_1, L_Flag, X;
$f_4(x_1);$                               //  Line_1++;
$f_1(x_2, f_9(x_1,x_3),$                  //  if(L_Flag) then Line_1+=X else
  $f_2(f_8(f_7(x_2),x_1),$                //    if(!L_Flag&Line_1) then
    $f_3(f_9(x_1,x_3), f_6(x_2,0)));$     //    {Line_1+=X; L_Flag=0;};
$f_5(x_1);$                               // Line_1--;
```

Step 2 (equivalent transformation using axioms (1-1),(1-2))

```
$f_0(x_1,x_2,x_3);$                       // unsigned Line_1, L_Flag, X;
$f_4(x_1);\; f_6(x_2,0);$                 //  Line_1++; L_Flag=0;
$f_1(x_2, f_9(x_1,x_3),$                  //  if(L_Flag) then Line_1+=X else
  $f_2(f_8(f_7(x_2),x_1),$                //    if(!L_Flag&Line_1) then
    $f_3(f_9(x_1,x_3), f_6(x_2,0)));$     //    {Line_1+=X; L_Flag=0;};
$f_5(x_1);$                               // Line_1--;
```

Step 3 (equivalent transformation using Theorem (1-3))

```
$f_0(x_1,x_2,x_3);$                       // unsigned Line_1, L_Flag, X;
$f_4(x_1);\; f_6(x_2,0);$                 //  Line_1++; L_Flag=0;
  $f_2(f_8(f_7(x_2),x_1),$                //    if(!L_Flag&Line_1) then
    $f_3(f_9(x_1,x_3), f_6(x_2,0)));$     //    {Line_1+=X; L_Flag=0;};
$f_5(x_1);$                               // Line_1--;
```

Step 4 (equivalent conversion using axioms (1-4)-(1-7))

$f_0(x_1, x_2, x_3);$ // unsigned Line_1, L_ Flag, X;

$f_4(x_1);\ f_6(x_2, 0);$ // Line_1 + +; L_ Flag = 0;

$f_9(x_1, x_3); f_6(x_2, 0);$ // Line_1+ = X; L_ Flag = 0;

$f_5(x_1);$ // Line_1 - -;

Step 5 (transformation using axioms (1-1), (1-2), (1-7),(1-8))

$f_0(x_1, x_2, x_3);$ // unsigned Line_1, L_ Flag, X;

$f_9(x_1, x_3);$ // Line_1+ = X;

Which is equivalent to Program 1-2.

Typically, such a noticeable optimization of a small fragment of a program indicates a logical error in the program (or a typo that did not lead to a syntax error). In this case, it is advisable for the optimizer program to issue a corresponding warning.

Example for Lisp language constructs

Lisp language constructs are represented by the following algebraic system:

$F_{Lisp} = < A_{Lisp}, \Omega_{Lisp}, R_{Lisp} >,$

where ALisp is a set of lists representing programs in the Lisp language.Ω_{Lisp}= {°, + }; - a set of operations consisting, respectively, of substitution operations and concatenation operations. Lots of Relations in Lisp Programs $R_{Lisp} = \{\ =,\ \neq,\ \perp\ \}$ consists respectively of the relations of equality, inequality and orthogonality (independence) of programs.

For program constructions we introduce the following notation (see Table 2). Using this notation, consider the following example. Suppose we need to prove the equivalence of the two Lisp programs presented below.

Program 2-1

```
(define ((( sample ( lambda ( x ) ( prog( mn ) ( setq mx )
( setq nx ) ( setq m ( append (car m)(cdr n))) (return m)))))))
```

Program 2-2

```
(define ((( sample ( lambda ( x ) ( prog( mn ) ( setq mx )
(setq nx) (return m)))))))
```

The following axioms and theorems are used for the proof.

(2-1) Axiom for introducing a known true conditional list:

$$(f_i(f_4(x_1, x_2), f_4(x_3, x_2)\, x_j) \ = \ (f_i(f_4(x_1, x_2), f_4(x_3, x_2)\, f_3(f_e(x_1, x_3))\, x_j)) \ .$$

(2-2) Theorem about invariant variables:

$$\{f_i(x_1, \ldots, x_n) \perp x_1, \ldots, x_n\} \ \& \ x_1 = x_2 \,\&\, x_2 = x_3 \,\&\ldots\&\, x_{n-1} = x_n \ =>$$

$$f_i(x_1, \ldots, x_n) = f_i(x_1, \ldots, x_1).$$

(2-3) Axiom of inverse list operations:

$$f_9(f_0(x), f_1(x)) = x \ .$$

(2-4) Axiom of empty assignment:

$$f_4(x, x) = 1 \ (\text{NULL}) \ .$$

Let us further write Program 2-1 in algebraic notation:

Step 1 (initial program structure)

$f_8(f_s(f_6(x, f_5(x_m, x_n), f_4(x_m, x), f_4(x_n, x)\, f_4(x_m, f_9(f_0(x_m), f_1(x_n))), f_7(x_m)))),$

where fs is the newly defined sample function.

Step 2 (equivalent transformation according to axiom (2-1))

$$f_8(f_s(f_6(x, f_5(x_m, x_n), f_4(x_m, x), f_4(x_n, x)\, f_3(f_e(x_n, x_m),$$
$$(f_4(x_m, f_9(f_0(x_m), f_1(x_n)))), f_7(x_m))))$$

or in Lisp syntax:

```
(define ((( sample ( lambda ( x ) ( prog( mn ) ( setq mx )
( setq nx (cond(equal(mn) ( setq m ( append (car m)(cdr n))))
(return m))))))) .
```

Step 3 (transformation according to theorem (2-2) and axiom (2-1):

$$f_8(f_s(f_6(x, f_5(x_m, x_n), f_4(x_m, x), f_4(x_n, x),$$
$$(f_4(x_m, f_9(f_0(x_n), f_1(x_n)))), f_7(x_m))))$$

or in Lisp syntax:

```
(define ((( sample ( lambda ( x ) ( prog( mn )
( setq mx ) ( setq nx ) ( setq m ( append (car n)(cdr n))) (return m))))))) .
```

Step 4 (transformation according to axioms (2-3, 2-4):

$$f_8(f_s(f_6(x, f_5(x_m, x_n), f_4(x_m, x), f_4(x_n, x)\, f_7(x_m)))).$$

Which corresponds to the text of Program 2-2.

Prolog Constructs

There is apparently no need to give examples of Prolog programs, since the equivalent transformations for this language are transformations of the well-known algebra of logic.

Interpretation of language constructs

Consideration of the interpretation of PL constructions makes it possible to explore their semantics, as well as mathematically strictly substantiate the axiomatics of PL PM.

Table 2

Notation of Lisp constructs in the form of program terms

N p / p	***Design designation in a Lisp program***	***Designation in the form software term***
1	(quote x1)	f0(x1)

2	(car x1)	f1(x1)
3	(cdr x1)	f2(x1)
4	(setq x1 x2)	f3 (x1, x2)
5	(cond x1 x2)	f4 (x1, x2)
6	(prog x1...xn)	f5 (x1, ..., xn)
7	(lambda x1 ... xn)	f6 (x1, ..., xn)
8	(return x1)	f7 (x1)
9	(define x1)	f8(x1)
10	(append x1 x2)	f9 (x1, x2)
eleven	(equal x1 x2)	fe (x1, x2)

Interpretation of syntactic constructions for the C language

As follows from paragraph 1.3, the interpretation of control syntactic structures for the C language is given by the following algebraic system:

$$\Phi_c = < A_c, \Omega_c, R_c > .$$

For Fs, all signature and relationship agreements are preserved, i.e. Ω_{With}=$\Omega_f$and Rc = Rф, but its own basis is used A_C^T for Ac carrier.

In the C language, as in many other universal programming languages, controls include four main structured programming constructs [37,136]:

- linear sequence of language operators,
- conditional full and abbreviated operators,
- loop construction with precondition,
- loop construction with postcondition.

The interpretation of these constructions is a subset of the basis A_C^T and is given by the derived elements Ac as follows.

Further in the text, the superscript of the functional interpretation symbols replaces the operation label, and the composition symbol is interpreted as a union of sets.

<u>*Linear sequence of statements*</u>

$f_1; f_2; \ldots; f_n;$- an expression in the syntax of the C language.

$\varphi_0^0(m(f_1), \varphi_0^1)\ o\ \varphi_0^1(m(\varphi_2), \varphi_0^2)\alpha..\alpha\varphi_0^{n-1}(m(f_n), stop);$ - interpretive sequence.

<u>*Conditional operator*</u>

if (f_1) then f_2 else f_3; - in the syntax of the language,

$\varphi_1^0(m(f_1),\varphi_2^1)\,o\,\varphi_2^1(m(\varphi_0^2),m(\varphi_0^3))\,o\,\varphi_0^2(m(f_2),stop)\,o\,\varphi_0^3(m(f_3),stop)$ - interpretive sequence.

Loop with precondition

while (f_1) f_2; - in the syntax of the C language,

$\varphi_1^0(m(f_1),\varphi_2^1)\,o\,\varphi_2^1(m(\varphi_0^2),stop)\,o\,\varphi_0^2(m(f_2),\varphi_1^0)$; - interpretive sequence.

Loop with postcondition

do f_1 while f_2; - in the syntax of the C language,

$\varphi_1^0(m(f_1),\varphi_1^1)\,o\,\varphi_1^1(m(f_2),\varphi_2^2)\,o\,\varphi_2^2(m(\varphi_0^1),stop)$; - interpretive sequence.

The interpretation of Ac also includes the interpretation of other syntactic constructions of the C language, which can be called applied, as well as the interpretation of all constructions built on them using the composition operation.

The algebraic interpretation system has its own axioms that characterize the properties of the operation of composition of various components of the carrier set.

(3-1) Commutativity axiom for independent linear composition:

$$m(f_1)\perp m(f_2) \Rightarrow \varphi_0^1(m(f_1),\varphi_0^2)\,o\,\varphi_0^2(m(f_2),\ stop) =$$
$$\varphi_0^2(m(f_2),\varphi_0^1)\,o\,\varphi_0^1(m(f_1),\ stop)\ .$$

(3-2) Commutativity axiom for independent terminal constructions:

$$m(\varphi_1)\perp m(\varphi_2) \Rightarrow \varphi_0^1(m(\varphi_1),\varphi_0^2)\,o\,\varphi_0^2(m(\varphi_2),\ stop) =$$
$$\varphi_0^2(m(\varphi_2),\varphi_0^1)\,o\,\varphi_0^1(m(\varphi_1),\ stop)\ .$$

(3-3) Axiom on the interpretation of reinitialization:

$I_1^=(m(x_i),I_2^=)\,o\,I_2^=(m(x_i),stop) \ = \ I_2^=(m(x_i),stop)$,

Where $I_i^=$ - initialization operation (an operation of changing the contents of memory m(xi) regardless of the previous contents of m(xi), assuming that m(xi) is not contained in the program code area).

The axioms of the algebraic system of interpretation are more visible and reliable than the axioms of the algebra of syntactic constructions of the language, since they operate with simpler concepts. In general, the idea of a “tower of languages” allows us to reduce any well-constructed interpretation system to a formal Turing machine.

In particular, consideration of the interpretation of C language constructions allows us to substantiate the reliability of the Fc axioms. So, for example, for axiom (1-1) from clause 1.4, the following proof can be given.

Step 1 (by definition of operator f0)

$$I(f_0(x_i,\ldots,x_n)) = \varphi_n^1(m(x_i),I_n^=)\,o\,I_1^=(m(x_i),I_2^=)o\ldots$$
$$o\varphi_n^n(m(x_n),I_1^=)\,o\,I_n^=(m(x_n),stop).$$

Step 2 (as defined by the f6 operator)

$$I(f_6(x_i,0)) = I_1^=(m(x_i,stop).$$

Step 3 (for the right side of axiom (1-1))

$$I(f_0(x_i,\ldots,x_n); f_6(x_i,0);\ldots f_6(x_n,0);) = \varphi_n^1(m(x_i),\varphi_1^{=})o\ldots o\varphi_n^n(m(x_i), I_1^{=})\ o$$

$$I_1^{=}(m(x_i), I_2^{=})o\ldots oI_n^{=}(m(x_n), I_{n+1}^{=})\ o\, I_{n+1}^{=}(m(x_i), I_{n+2}^{=})o\ldots oI_{n+n}^{=}(m(x_n), stop)\quad .$$

Step 4 (according to axioms (3-1) and (3-3))

$$\varphi_n^1(m(x_i),\varphi_1^{=})o\ldots o\varphi_n^n(m(x_i), I_1^{=})\ o\, I_1^{=}(m(x_i), I_2^{=})o\ldots$$

$$oI_n^{=}(m(x_n), I_{n+1}^{=})\ o\, I_{n+1}^{=}(m(x_i), I_{n+2}^{=})o\ldots oI_{n+n}^{=}(m(x_n), stop)\quad =$$

$$\varphi_n^1(m(x_i),\varphi_1^{=})o\ldots o\varphi_n^n(m(x_n),\varphi_1^{=})\ o\,\varphi_1^{=}(m(x_i), I_2^{=})o\ldots oI_n^{=}(m(x_n), stop).$$

Which corresponds to the interpretation I(f0(xi,...,xn)).

Interpretation of syntactic constructs for the Lisp language

The algebraic system describing the interpretation of Lisp constructions is given by the triple:

$$\Phi_{Lisp} =< A_{Lisp}, \Omega_{Lisp}, R_{Lisp} >,$$

where are the setsΩ_{Lisp}, RLisp are analogues of the corresponding setsΩ_f, Rf.

As a functional language, base Lisp uses a functional interpretation of almost all of its constructs:

$(f_0, f_1, \ldots, f_n)$ - in Lisp syntax,

$$\varphi_o^o(m(f_1),\varphi_o^1)\ o\,\varphi_o^1(m(f_2),\varphi_o^2)o\ldots o\varphi_o^n(m(f_n),\varphi_o^{n+1})\ o$$

$$o\varphi_1^{n+1}(m(\varphi_o^o) + m(\varphi_o^1)+\ldots+m(\varphi_o^n),\ stop)$$

- in interpretation.

The relative ease of interpretation of Lisp control syntactic structures predetermines the justification of most Lisp axioms within the framework of an applied interpretation system.

So, for example, in the applied Lisp interpretation system, one can distinguish the interpretation of Lisp functions: quote (saving an argument), append (combining arguments), car (issuing the first element of a complex list argument), cdr (issuing all elements of the argument except the first).

$$I^{a\,ppend}(m_1 + m_2,\ stop)\ =\ I^{quote}(m(m_1 + m_2),\ stop),$$

$$I^{car}(m_1 + m_2 + \ldots + m_n, stop) = I^{quote}(m_1, stop),$$

$$I^{cdr}(m_1 + m_2 + \ldots + m_n, stop) = I^{quote}(m(m_2 + \ldots + m_n), stop).$$

Based on these relationships and the interpretation of control constructs of the Lisp language, one can, for example, justify axiom (3-3):

(append (car x)(cdr x)) = (quote x).

Step 1 (recording the interpretation of the left side of the axiom)

$$I_o^{car}(m_1 + m_2 + \ldots + m_n, I_1^{cdr})\ o\, I_1^{cdr}(m_1 + m_2 + \ldots + m_n, I_2^{append})\ o$$

$$o\, I_2^{append}(m(I_0^{car}) + m(I_0^{cdr}), stop)\ .$$

Step 2 (from interpretive definitions)

$$m(I_0^{car}) = m_1,\ m(I_1^{cdr}) = m_2 + m_3 + \ldots + m_n,$$

$$I_2^{append}(m_1 + m_2 + \ldots + m_n,\ stop) = I^{quote}(m(m_1 + m_2 + \ldots + m_n), stop),$$

which corresponds to the interpretation (quote x).

Constructs for the Prolog language

The interpretation of syntactic constructions in logical languages has been well studied within the framework of the model-theoretic approach in the study of the interpretation of applied first-order theories into algebraic systems.

Thus, algebraic formalization allows us to study the PMs of various algorithmic languages from a unified point of view, and therefore makes it possible to compare these PMs, develop new ones, and describe the semantics of the LP in algebraic notation.

The algebraic systems that make up the PM can be used to study the PL both separately and in combination, within the framework of a multi-basic algebraic system. The construction of such a system for any language or other instrumental software, in our opinion, is advisable for studying issues of optimization, automatic (automated program construction), as well as studying the functional integrity of software systems based on language formalization.