UDC 621.398
https://doi.org/10.32362/2500-316X-2022-10-1-18-27

RESEARCH ARTICLE

Equivalence of the schemes of programs based on the algebraic approach to setting the semantics of programming languages

Yuri P. Korablin @

NRU MPEI, Moscow, 111250 Russia
@ *Corresponding author, e-mail: KorablinYP@mpei.ru*

Abstract
Objectives. The paper deals with the equivalence of program schemes. According to A.A. Lyapunov and Yu.I. Yanov, the founders of this theory, a program scheme is understood as a program model wherein abstraction from contensive values of operators and expressions is performed. In this case, the program structure containing symbolic notation of operators and expressions remains unchanged while maintaining their execution sequence. The programming language model presented in the paper contains basic constructs of sequential languages and is the core of the existing sequential programming languages. The paper aimed at developing an effective algorithm for studying equivalence (nonequivalence) of program schemes of sequential programming languages.
Methods. An algebraic approach to specifying semantics of programming languages was used for studying the equivalence of program schemes.
Results. A process semantics being the new algebraic approach to specifying the formal semantics of sequential programming languages was proposed. The process semantics was specified by matching programs (program schemes) with a set of computation sequences. The computation sequence was understood as the execution sequence of actions (commands and tests) of the program. Two types of concatenation operations (test–command and command–command) and the merge operation, which properties are given by axiomatic systems, were defined in the introduced semantic domain. The finiteness of the semantic value representation in the form of systems of recursive equations was proved. The algorithm for proving the equivalence (nonequivalence) of systems of recursive equations characterizing semantic values for a pair of program schemes was proposed, which implies the equivalence (nonequivalence) of programs in the strong sense.
Conclusions. The paper demonstrates the efficient use of the proposed algorithm for proving the equivalence of sequential program schemes excluding side effects when calculating expressions, i.e., sequential computation of the expression more than once does not change anything. The example of proving the equivalence of program schemes by two methods—the well-known de Bakker–Scott fixed-point induction method and the method proposed by the author—is given. Comparison of the above methods testifies not only to the new method's effectiveness but also to its significant simplicity, proved in practice by students who performed corresponding tasks when studying the Semantics of Programming Languages at the Institute of Information and Computing Technologies at the National Research University Moscow Power Engineering Institute (Moscow, Russia).

Keywords: program scheme, semantic domains, process semantics, equational characterization of the semantic meanings of programs, equivalence of program schemes

• **Submitted:** 15.06.2021 • **Revised:** 29.09.2021 • **Accepted:** 27.12.2021

For citation: Korablin Yu.P. Equivalence of the schemes of programs based on the algebraic approach to setting the semantics of programming languages. *Russ. Technol. J.* 2022;10(1):18–27. https://doi.org/10.32362/2500-316X-2022-10-1-18-27

Financial disclosure: The author has no a financial or property interest in any material or method mentioned.

The author declares no conflicts of interest.

НАУЧНАЯ СТАТЬЯ

Эквивалентность схем программ на основе алгебраического подхода к заданию семантики языков программирования

Ю.П. Кораблин @

НИУ МЭИ, Москва, 111250 Россия
@ Автор для переписки, e-mail: KorablinYP@mpei.ru

Резюме
Цели. Статья посвящена вопросам эквивалентности схем программ. В соответствии с работами А.А. Ляпунова и Ю.И. Янова – основоположников данной теории, под схемой программы понимается ее модель, в которой осуществляется абстрагирование от содержательных значений операторов и выражений. При этом неизменной остается структура программы, включающая символические обозначения операторов и выражений при сохранении последовательности их выполнения. Представленная в статье модель языка программирования содержит основные конструкции последовательных языков и является ядром имеющихся языков последовательного программирования. Цель работы – разработка эффективного алгоритма исследования вопросов эквивалентности (неэквивалентности) схем программ последовательных языков программирования.
Методы. Используется алгебраический подход к заданию семантики языков программирования для исследования вопросов эквивалентности схем программ.
Результаты. Предложен новый алгебраический подход к заданию формальной семантики языков последовательного программирования – процессная семантика. Процессная семантика задается посредством сопоставления программам (схемам программ) множества вычислительных последовательностей. Под вычислительной последовательностью понимается последовательность выполнения действий (команд и тестов) программы. На введенной семантической области (множестве вычислительных последовательностей) определены операции конкатенации двух видов (тест-команда и команда-команда) и операция объединения, свойства которых заданы системами аксиом. Доказана конечность представления семантических значений в виде систем рекурсивных уравнений. Предложен алгоритм доказательства эквивалентности (неэквивалентности) систем рекурсивных уравнений, характеризующих семантические значения для пары схем программ, откуда вытекает эквивалентность (неэквивалентности) программ в сильном смысле.
Выводы. Показана эффективность применения предложенного алгоритма для доказательства эквивалентности схем последовательных программ, в которых отсутствует побочный эффект при вычислении выражений, т.е. последовательное вычисление выражения более, чем один раз, ничего не меняет. В статье приведен демонстрационный пример доказательства эквивалентности схем программ двумя методами: известным методом индукции фиксированной точки де Баккера – Скотта и предложенным в статье методом. Сравнение приведенных методов свидетельствует не только об эффективности нового метода, но и его существенной простоте, что было подтверждено на практике при выполнении соответствующих заданий студентами специальности «Прикладная математика и информатика» Национального исследовательского университета МЭИ в процессе изучения дисциплины «Семантика языков программирования».

Ключевые слова: схема программы, семантические области, процессная семантика, эквациональная характеризация семантических значений программ, эквивалентность схем программ

• **Поступила:** 15.06.2021 • **Доработана:** 29.09.2021 • **Принята к опубликованию:** 27.12.2021

Для цитирования: Кораблин Ю.П. Эквивалентность схем программ на основе алгебраического подхода к заданию семантики языков программирования. *Russ. Technol. J.* 2022;10(1):18–27. https://doi.org/10.32362/2500-316X-2022-10-1-18-27

Прозрачность финансовой деятельности: Автор не имеет финансовой заинтересованности в представленных материалах или методах.

Автор заявляет об отсутствии конфликта интересов.

INTRODUCTION

Program equivalence is an extremely important aspect of the theory and practice of programming languages underlying such problems as program correctness [1–10], program completeness (incompleteness), and equivalent transformations of programs for optimizing them [11–13] in one way or another. It is clear that to solve these problems, appropriate formal methods for specifying semantics of programming languages should be developed. Propositional logic-based methods have been among the first to be developed, the most known being Floyd's inductive assertion method [5] and Hoare's axiomatic method [6]. The use of these methods allows proving partial correctness and completeness (incompleteness) of a sufficiently large class of programs limited in size. The denotational approach [7] allows more opportunities for solving the problem of program equivalence using methods based on fixed point properties, in particular, the de Bakker–Scott fixed-point induction method [4, 7]. In the paper, the algebraic method for specifying the process semantics of programs, matching programs with a set of computation sequences (execution paths) of a program as semantic values, is proposed. The algorithm for analyzing the equivalence of program schemes based on the idea of representing semantic values in the form of finite systems of recursive equations with further analysis for equivalence (non-equivalence) of the obtained systems of recursive equations developed by the author in his thesis[1] and [14] is proposed. In contrast to the method proposed for proving the equivalence of program schemes[1], the recursive equations obtained in the systems have a more complex form and require more detailed analysis. The effectiveness of this method for proving the program scheme equivalence is shown.

1. FORMAL MODEL OF PROGRAMMING LANGUAGE

We shall specify the syntax and semantics of language L used as a programming language model.

[1] Korablin Yu.P. Semantic methods of analysis of distributed systems. Dr. Thesis (Eng.). Moscow: MEI; 1994. 40 p. (in Russ.).

1.1. Syntax of Language L

The alphabet of the language consists of the following:

- the set of elementary commands Com with typical element a;
- the set of Boolean expressions Exp with typical element b.

Typical elements of sets may be indexed. In addition, the following three constants may be defined: skip is empty command; tt and ff are identically true and identically false Boolean values, respectively.

The set of Cmd commands with a typical element c may be defined as follows:

$$c ::= \text{skip} \mid a \mid [gc] \mid \star\ [gc]\ ,$$

where gc is a typical element of the set of protected Gcom commands with the following syntax:

$$gc ::= g \rightarrow c \mid gc\ \{\ \square\ gc\ \},$$
$$g ::= tt \mid ff \mid b$$

Parentheses are used to denote zero or more iterations of the construct enclosed in brackets. A typical element of the set of protections G is denoted by g.

The symbol □ connects alternative program components, i.e., such constructs are analogous to switch branching statement in programming languages.

The set of programs with typical element pr may be defined as follows:

pr ::= c | pr; c.

1.2. Algebraic semantics: mathematical foundations and semantic equalities

The semantic function matches the program written in L language with a set of computation sequences (CS) that may be used for executing that program.

To specify the program semantics, the principle of constructing semantic value of the entire program based on semantic values of that program components is proposed.

Assuming that any program may be, in turn, a component of another program, *a priori* semantics of the entire program may always be obtained by composing semantic values of program components.

1.2.1. Semantic domains

1. The set of values of elementary commands ACom with typical element A.

2. The set of values of Boolean expressions Bexp with typical element B.

3. The Const set containing the constants τ (identity transformation), T (identically true value), and F (identically false value).

Typical elements of semantic domains may be indexed.

We shall define the set of computation sequences CPath with typical element cp.

Two additional sets being the Test set with typical element β and the Action set with typical element α may be predefined, as follows:

$\beta ::= T \mid F \mid B,$
$\alpha ::= \tau \mid A,$
$cp ::= \alpha \mid \beta \wedge cp \mid cp1 \circ cp2,$

where $cp*$ is finite CS while $cp\omega$ is infinite CS. We shall introduce the set $SP = \mathcal{P}(CPath)$ with typical element sp, i.e., SP is the power set defined as the set of all subsets of the CPath set.

Then, operations of the set–theoretic union (sp1 + sp2), sequential composition (sp1 $\circ$ sp2), and the least fixed point (sp^+) are defined in the SP set.

Before determining the value for sp^+, two auxiliary definitions should be introduced.

Definition 1. CS cp is called empty and denoted by ε if:

1) $cp \equiv \tau$;

2) $cp = T \wedge \varepsilon$.

Infinite sequence of the form $\varepsilon\omega$ is denoted by LOOP (looping).

Definition 2. $\varepsilon \in sp$ if:

$sp \equiv \varepsilon$;

sp = sp1 + sp2 and ε belongs to at least one of the sets sp1 or sp2;

sp = sp1 $\circ$ sp2 and ε belongs to both sp1 and sp2 simultaneously.

We shall define sp^+ as the least fixed point of the operator F(sp), i.e., $sp^+ = \mu F(sp)$, where F(sp) may be written in the following form:

$F(sp) = \lambda q.\ sp \circ q + \nu$, where $\nu = \tau$.

1.2.2. Semantics of language L

The semantic function may be defined as follows:

$\mathbb{C}[\![skip]\!] = \tau,$
$\mathbb{C}[\![a]\!] = A,$
$\mathbb{C}[\![tt \rightarrow pr]\!] = \mathbb{E}[\![tt]\!] \wedge \mathbb{C}[\![pr]\!],$
$\mathbb{C}[\![ff \rightarrow pr]\!] = \mathbb{E}[\![ff]\!] \wedge \mathbb{C}[\![pr]\!],$
$\mathbb{C}[\![b \rightarrow pr]\!] = \mathbb{E}[\![b]\!] \wedge \mathbb{C}[\![pr]\!],$

where function $\mathbb{E}$: Exp $\rightarrow$ TEST defines semantic values of the expressions.

$\mathbb{C}[\![*\{gc\}]\!] = (\mathbb{C}[\![gc]\!])^+,$
$\mathbb{C}[\![gc1 \square gc2]\!] = \mathbb{C}[\![gc1]\!] + \mathbb{C}[\![gc2]\!],$
$\mathbb{C}[\![pr1; pr2]\!] = \mathbb{C}[\![pr1]\!] \circ \mathbb{C}[\![pr2]\!].$

Then, the semantic function for the following expressions may be defined:

$\mathbb{E}$: Exp $\rightarrow$ TEST,
$\mathbb{E}[\![tt]\!] = T,$
$\mathbb{E}[\![ff]\!] = F,$
$\mathbb{E}[\![b]\!] = B.$

1.3. Axiomatization of semantic domain properties

The properties of operations on elements of the semantic domain $SP\sigma = \mathcal{P}(CPath\sigma)$ may be described using an axiomatic system (axiom schemes) and inference rules. The axiomatic system may be defined as a set of blocks, each describing certain properties of semantic objects.

The ACTION $\cup$ TEST set with typical element d′ may be denoted by D′. Metavariables X, Y, and Z may be used for denoting elements of the semantic domain.

Axioms defining basic properties of operations "$\circ$", "$\wedge$" and "+" may be written as follows:

(A1) $X + X = X,$
(A2) $X + Y = Y + X,$
(A3) $X + (Y + Z) = (X + Y) + Z,$
(A4) $(X + Y) \circ Z = X \circ Z + Y \circ Z,$
(A5) $(X \circ Y) \circ Z = X \circ (Y \circ Z),$
(A6) $\tau \circ X = X,$
(A7) $X \circ \tau = X,$
(A8) $LOOP \circ X = LOOP,$
(A9) $\varnothing \circ X = \varnothing,$
(A10) $X + \varnothing = X,$
(A11) $(\beta \wedge X) \circ Y = \beta \wedge (X \circ Y),$
(A12) $F \wedge X = \varnothing,$
(A13) $\varnothing \wedge X = \varnothing.$

2. EQUATIONAL CHARACTERIZATION OF *A PRIORI* SEMANTIC VALUES

In this section, the possibility of representing the program semantic values in the form of finite systems of recursive equations is described.

Definition 3. Let $\alpha \in ACT = ACTION \setminus \{\tau, \varnothing\}$ and $(\beta \in TES = TEST \setminus \{F, \varnothing\}$ and $PREF = ACT \cup TES$.

The partial functions prefix: SP → PREF and suffix: SP → SP may be defined using Table 1:

Table 1. Defining prefixes and suffixes of computation sequences (sp)

sp	prefix (sp)	suffix (sp)
$\alpha \circ X$	α	X
$\tau \circ X$	prefix (X)	suffix (X)
$\beta \wedge X$	β	X

It should be noted that prefix and suffix may be not defined for the expression sp ≡ F ^ X, since F ^ X ≡ Ø (axiom A12).

Definition 4. The CS set of P ∈ SPσ may be equationally characterized if there exists the finite set P1, P2, ..., Pn ∈ SP such that P ≡ P1 for any i $(1 \le i \le n)$:

$$P_i = \sum_{j\in N} \alpha_{ij} \circ P_{ij} + \sum_{k\in N} \beta_{ik} \wedge P_{ik} + \delta(P_i), \qquad (*)$$

where $N = \{1, 2, \ldots, n\}$, $\alpha_{ij} \in$ ACT, $\beta_{ik} \in$ TES,

$\forall i$ δ(Pi) ⊂ $\mathcal{P}$(ACT′ ∪ {LOOP}), where ACT′ = ACTION \ {τ},

$\forall i\ \forall j\ \exists l \in N$, that P$ij$ = Pl,

$\forall i\ \forall k\ \exists r \in N$, that P$ik$ = Pr.

Theorem 1. Every SP set matched as a semantic value with a program may be equationally characterized using a finite system of equations of the form (*).

Proof. The proof is by induction method for structure P.

Basis. The equational characterization for p ≡ d′, where d′ ∈ D′ follows trivially.

Inductive step. Let P1 = SP and P2 = SP be equationally characterizable. Then it is necessary to prove that P1 ∘ P2, β ^ P1, P1 + P2, and P1$^+$ are equationally characterizable.

We shall prove the expression P = P1 ∘ P2.

By the induction hypothesis, there exist sets P11, P12, ..., P1n and P21, P22, ... P2m such that P1 ≡ P11 and P2 ≡ P21, and

$$\mathrm{P1}i = \sum_{j\in N} \alpha_{ij} \circ \mathrm{P1}_{ij} + \sum_{k\in N} \beta_{ik} \wedge \mathrm{P1}_{ik} + \delta(\mathrm{P1}i),\ i = \overline{1,n}, \quad (**)$$

and

$$\mathrm{P2}i = \sum_{r\in N} \alpha_{ir} \circ \mathrm{P2}_{ir} + \sum_{p\in N} \beta_{ip} \wedge \mathrm{P2}_{ip} + \delta(\mathrm{P2}i),\ i = \overline{1,m}, \quad (***)$$

We shall denote

$$\eta(u, v1, \ldots, vr) = \mathrm{P1}_u \circ \mathrm{P2} + \mathrm{P2}_{v_i} + \ldots + \mathrm{P2}_{v_r}, \qquad (1)$$
$$(u = \overline{0,n},\ r \ge 0,\ 1 \le vi \le m,\ i = \overline{1,r}).$$

We shall write $\mathrm{P10} \circ \mathrm{P2} + \mathrm{P2}_{v_i} + \ldots + \mathrm{P2}_{v_r}$ instead of $\mathrm{P2}_{v_i} + \ldots + \mathrm{P2}_{v_r}$.

The number of expressions (1) is finite. By the inductive hypothesis, the following may be written:

$$\eta(u, v1, \ldots, vr) =$$
$$= \left(\sum_{j\in N} \alpha_{uj} \circ \mathrm{P1}_{uj} + \sum_{k\in N} \beta_{uk} \wedge \mathrm{P1}_{uk} + \delta(\mathrm{P1}u) \right) \circ \mathrm{P2} +$$
$$+ \sum_{j\in N} \alpha_{v_1 j} \circ \mathrm{P2}_{v_1 j} + \sum_{k\in N} \beta_{v_1 k} \wedge \mathrm{P2}_{v_1 k} + \delta(\mathrm{P2}v_1) + \ldots +$$
$$+ \sum_{j\in N} \alpha_{v_r j} \circ \mathrm{P2}_{v_r j} + \sum_{k\in N} \beta_{v_r k} \wedge \mathrm{P2}_{v_r k} + \delta(\mathrm{P2}v_r).$$

Then, applying axioms A1, A2, A3, and A5, the following may be written:

$$\eta(u, v1, \ldots, vr) =$$
$$= \sum_{j\in N} \alpha_{uj} \circ \mathrm{P1}_{uj} \circ \mathrm{P2} + \sum_{k\in N} \beta_{uk} \wedge (\mathrm{P1}_{uk} \circ \mathrm{P2}) +$$
$$+ \sum_{j\in N} \alpha_{v_1 j} \circ \mathrm{P2}_{v_1 j} + \sum_{k\in N} \beta_{v_1 k} \wedge \mathrm{P2}_{v_1 k} + \ldots +$$
$$+ \sum_{j\in N} \alpha_{v_r j} \circ \mathrm{P2}_{v_r j} + \sum_{k\in N} \beta_{v_r k} \wedge \mathrm{P2}_{v_r k} +$$
$$+ \delta(\mathrm{P1}u) \circ \mathrm{P2} + \delta(\mathrm{P2}v_1) + \ldots + \delta(\mathrm{P2}v_r).$$

If $\delta(\mathrm{P1}u) = \sum_{j\in N} \delta_{ui}$, where $\delta\mathrm{ui} \in \mathrm{D}'$, then axiom A4 is applied again replacing summand δ(P1u) ∘ P2 with expression $\sum_{j\in N} \delta_{ui} \circ \mathrm{P2}$, in η($u$, v1, ..., vr), and if $\delta_{ui} = \tau$, then P2 is replaced by its representation for P21 from (***).

Then, applying axioms A1, A2, A3, and A5, the following may be written:

$$\eta(u, v1, \ldots, vr) = \sum_{j\in N} \alpha_{uj} \circ \mathrm{P}_{uj} + \sum_{k\in N} \beta_{uk} \wedge \mathrm{P}_{uk} +$$
$$+ \sum_{q\in N} \sum_{j\in N} \alpha_{qj} \circ \mathrm{P}_{qj} + \sum_{q\in N} \sum_{k\in N} \beta_{qk} \wedge \mathrm{P}_{qk} +$$
$$+ \delta(u, v1, \ldots, vr),$$

where all expressions $\mathrm{P}_{uj}, \mathrm{P}_{uk}, \mathrm{P}_{qj}$, and P_{qk} are included in (1). Since η(1,1) ≡ P1 ∘ P2, then P1 ∘ P2 is equationally characterizable.

P = β ^ P1. This is a trivial case. In this case, only one equation P = P1 = β ^ P11 is added to the finite system of equations for P1 = P11, which implies the equational characterizability of the expression β ^ P1.

Proving the equational characterization of the expression P = P1 + P2 is analogous, where the following set is used as a finite set:

$$\xi(u,v) \equiv \mathrm{P1}u + \mathrm{P2}v,\ u = \overline{0,n};\ v = \overline{0,m}. \qquad (2)$$

For proving the equational characterization of the expression $P = P1^{+}$, the set

$$\zeta(u1,...,ur) \equiv (\mathrm{P1}u_1 + ... + \mathrm{P1}u_r) \circ \mathrm{P1}^{+},$$
$$r \geq 0,\ 1 \leq ui \leq n,\ i = \overline{1,r}. \qquad (3)$$

may be used.

The number of expressions (3) is finite.

Thus, the set of execution paths for a program, i.e., process semantics allowing studying various program properties, may be defined as the semantic value of a program. In particular, as described below, this approach allows studying the equivalence of program schemes in a strong sense.

3. COMPARISON OF PROMRAM SCHEMES IN LANGUAGE L

We shall define a method for comparing systems of recursive equations of the form (*), thus allowing obtaining a formal method for comparing program schemes in language L.

The significant factor of the considered method is the uniqueness of prefixes of any recursive equation being achieved through applying the axiom of the form X ∘ (Y + Z) = X ∘ Y + X ∘ Z. The result of applying this axiom is that any recursive equation is reduced to the form where all α_{ij} are pairwise distinct.

Thus, proving the equivalence (or non-equivalence) of two systems of recursive equations in each step is reduced to proving the equivalence of expression pairs having the same prefixes in the considered recursive equations.

The comparison process ends when new expression pairs stop appearing, or there is a mismatch of prefixes or expression sets δ for a certain expression pair in a certain step.

The first case of completing the comparison process implies the equivalence of two systems of recursive equations, while the second case implies their noncomparability, and therefore non-equivalence of two systems of recursive equations.

Let there be two systems of recursive equations of the form (*) to be tested for equivalence. These systems contain recursive equations for expressions P1, P2, …, P*n* и P1′, P2′, ..., P*m*′, respectively. We shall denote the set of expressions {P1, P2, ..., P*n*} by P, {P1′, P2′, ..., P*m*′} by P′, and the sets of all subsets of P and P′ by $\mathcal{P}$(P) and $\mathcal{P}$(P′), respectively.

We shall consider the equations for P1 and P1′. Here, the following cases are possible:

a) δ(P1) = δ(P1′) and $\forall j \in N\ \exists k \in N$ are such that $\alpha 1j \equiv \alpha 1k'$ and *vice versa*, and also $\forall k \in N\ \exists p \in N$ is such that $\beta 1k \equiv \beta 1p'$ and *vice versa*. In this case, the process of writing out equations for all pairs (P_{1j}, P_{1k}) having the same prefixes of type α as well as for all pairs (P_{1k}, P_{1p}) having the same prefixes of type β should be continued.

b) At least one of the conditions given in paragraph a) is not satisfied. This is the case of noncomparability of the prefix set or that of noncomparability of absolute terms of equations (δ(P1) ≠ δ(P1′)), whence it follows that computation sequence sets (CSSs) given by systems of recursive equations are noncomparable.

The above process of writing equations should be proceeded for the resulting pairs until obtaining one of the following results:

- two systems of recursive equations equivalent up to notations (with only case a) occurred in each step) are set up. In this case, CSSs given by systems of recursive equations are comparable, and thus, the program schemes matched with these expressions are equivalent;
- the condition given in paragraph a) is not satisfied. In this case, CSSs given by systems of recursive equations are not equivalent, and thus, the program schemes matched with these expressions are not equivalent.

We shall analyze the effectiveness of the proposed method for proving the equivalence of program schemes. In particular, an example of proving the following statement applying the extensively used method of fixed-point induction [4] may be firstly considered:

$\mathbb{C}$ ⟦while B do C1 od; while B do C2 od⟧ = $\mathbb{C}$ ⟦while B do C1 od⟧.

Solution. We shall introduce the following notations: $\mathbb{E}$ ⟦E⟧=ω; $\mathbb{C}$ ⟦C1⟧ = γ1; $\mathbb{C}$ ⟦C2⟧ = γ2.

$\mathbb{C}$ ⟦while E do C1 od;⟧ = fix(λγ.λσ.ωσ → γ·γ1σ,σ) = = fix H1 = γ1′,

where fix H1 stands for taking the least fixed point of the operator H1= λγ.λσ.ωσ → γ·γ1σ,σ.

$\mathbb{C}$ ⟦while E do C2 od;⟧ = fix(λγ.λσ.ωσ → γ·γ2σ,σ) = = fix H2 = γ2′,

where H2 = λγ.λσ.ωσ → γ·γ2σ,σ.

The left-hand side (LHS) of the equation is given as follows:

LHS = $\mathbb{C}$ ⟦while E do C2 od;⟧ · $\mathbb{C}$ ⟦while E do C1 od;⟧ = = $\gamma 2' \cdot \gamma 1'$.

The right-hand side (RHS) of the equation is, respectively, of the following form:

RHS = $\gamma 1'$.

Thus, it is required to prove that: $\gamma 2' \cdot \gamma 1' = \gamma 1'$.

For proving, the method of fixed-point induction is applied.

Let $q(x) \equiv \gamma 2' \cdot x = x$.

1. $q(\bot) \equiv \gamma 2' \cdot \bot = \bot$
2. Let $q(x) \equiv \gamma 2' \cdot x = x$ is valid.

We shall show that $q(H1(x)) \equiv \gamma 2' \cdot H1(x) = H1(x)$ is valid.

LHS = $\gamma 2' \cdot H1(x) = \gamma 2' \cdot (\lambda\sigma.\omega\sigma \rightarrow x \cdot \gamma 1\sigma, \sigma)$ = (left factoring)

= $\lambda\sigma.\omega\sigma \rightarrow \gamma 2' \cdot x \cdot \gamma 1\sigma,\ \gamma 2'\sigma$ = (assuming validity of $q(x)$)

= $\lambda\sigma.\omega\sigma \rightarrow x \cdot \gamma 1\sigma,\ \gamma 2'\sigma$ = (by the fixed point property)

= $\lambda\sigma.\omega\sigma \rightarrow x \cdot \gamma 1\sigma,\ H2(\gamma 2')\sigma$ =

= $\lambda\sigma.\omega\sigma \rightarrow x \cdot \gamma 1\sigma,\ (\lambda\sigma.\omega\sigma \rightarrow \gamma 2' \cdot \gamma 2\sigma, \sigma)\sigma$ =

= $\lambda\sigma.\omega\sigma \rightarrow x \cdot \gamma 1\sigma,\ (\lambda\sigma.\omega\sigma \rightarrow \gamma 2' \cdot \gamma 2\sigma, \sigma)$ = (by the conditional operator property)

= $\lambda\sigma.\omega\sigma \rightarrow x \cdot \gamma 1\sigma,\ \sigma$.

RHS = $H1(x) = \lambda\sigma.\omega\sigma \rightarrow x \cdot \gamma 1\sigma,\ \sigma$

It follows from paragraphs 1 and 2 that $q(\text{fix } H1) = q(\gamma 1')$ is valid, and thus,

$\gamma 2' \cdot \gamma 1' = \gamma 1'$, as required.

This statement way be written using the proposed notation, as follows:

*[b → c1]; *[b → c2] = *[b → c1].

Using the above algorithm, the validity of the following statement may be proved:

$\mathbb{C}$ ⟦ *[b → c1]; *[b → c2] ⟧ = $\mathbb{C}$ ⟦ *[b → c1] ⟧.

We shall denote $\mathbb{C}$ ⟦ *[b → c1]; *[b → c2] ⟧ by P1, and $\mathbb{C}$ ⟦ *[b → c1] ⟧ by P2.

P1 $= (B \wedge C1)^{+} \circ (B \wedge C2)^{+}$,
P2 $= (B \wedge C1)^{+}$.

Then, P1 and P2 may be represented by systems of recursive equations according to the program scheme comparison algorithm given above. Let P11 ≡ P1 and P21 ≡ P2. Then:

$P11 = \tau \circ (B \wedge C2)^{+} + (B \wedge C1) \circ (B \wedge C1)^{+} \circ (B \wedge C2)^{+} =$
$= \tau + (B \wedge C2) \circ (B \wedge C2)^{+} + (B \wedge C1) \circ (B \wedge C1)^{+} \circ$
$\circ (B \wedge C2)^{+} =$
$= \tau + B \wedge \{C2 \circ (B \wedge C2)^{+} + C1 \circ (B \wedge C1)^{+} \circ (B \wedge C2)^{+}\} =$
$= \tau + B \wedge P12$.
$P21 = \tau + (B \wedge C1) \circ (B \wedge C1)^{+} = \tau + B \wedge \{C1 \circ$
$\circ (B \wedge C1)^{+}\} = \tau + B \wedge P22$.

It can be easily seen that the set P12 contains CSs starting at C2 while the set P22 does not contain CSs starting at C2, whence it follows that P12 ≠ P22.

At first glance, this result seems contradicting what obtained above. However, the crux of the problem is that obtaining this result, the lack of side effects while calculating expressions (conditions for execution of protected commands) has been assumed. It follows from this assumption directly that recalculation of the same expression would always give the same result. In our case, however, recalculating the same expression may give, in general, different results. This may be illustrated by the following example. Let be a program enabling side effects, i.e., value changes in variables when calculating expressions,

x: = 1; *[x: = 2 × x = 4 → c1]; [x: = 2 × x = 4 → c2].

The initial calculation of the expression x: = 2 × x = 4 results in a false value, and thus in breaking out of the loop *[x: = 2 × x = 4 → c1] and in proceeding to calculations of the next instruction. In this case, the value of variable x before executing is equal to 2. Recalculating the expression x: = 2 × x = 4 results in the true value. Obviously, the presence of side effects is usually an undesirable situation. Therefore, it would be advisable to create a formalism allowing analyzing programs excluding side effects. To that end, a number of notions introduced previously should be redefined, as well as axioms characterizing their properties should be added.

We shall first expand the test set by adding negation and difficult tests.

$\beta ::= \ldots \mid \urcorner\beta \mid \beta 1 * \beta 2$, where the ellipsis stands for the predefined test sets.

The new test set is characterized by the following block of axioms:

(G1) $\beta 1 * \beta 2 = \beta 2 * \beta 1$,
(G2) $\beta * \urcorner\beta = F$,
(G3) $F * \beta = F$,
(G4) $\varnothing * \beta = \varnothing$,

(G5) $T * \beta = \beta$,
(G6) $\beta * \beta = \beta$,
(G7) $\beta 1 * (\beta 2 * \beta 3) = (\beta 1 * \beta 2) * \beta 3$,
(G8) $\beta 1 \wedge (\beta 2 \wedge C) = (\beta 1 * \beta 2) \wedge C$.

The expansion of multiple tests results in the need to modify the predefined notions of prefix and suffix.

Definition 5. The partial functions prefix: SP → PREF and suffix: SP → SP are defined by Table 2 being the extension of Table 1:

Table 2. Defining prefixes and suffixes of computation sequences (sp)

SP	prefix (SP)	suffix (SP)
$\alpha \circ X$	α	X
$\tau \circ X$	prefix (X)	suffix (X)
$\beta 1 \circ X1$, where $X1 \neq \beta 2 \wedge X2$, and $X1 \neq \tau \circ X$, and $X1 \neq X2^+$	$\beta 1$	X1
$\beta 1 \wedge (\beta 2 \wedge X)$	prefix $((\beta 1 * \beta 2) \wedge \wedge X)$	suffix $((\beta 1 * \beta 2) \wedge \wedge X)$
$\beta \wedge (\tau \circ X)$	prefix $(\beta \wedge X)$	suffix $(\beta \wedge X)$

It should be noted that when sp $\equiv \beta 1 \wedge X1$ and $X1 \equiv X2^+$, it would be essential to use first the property of the least fixed point up to the appearance of one of the constructs which the suffix and prefix defined for in Table 2.

Then, the following statement given above should be proved:

$$\mathbb{C} ⟦ *[b \rightarrow c1]; *[b \rightarrow c2] ⟧ = \mathbb{C} ⟦ *[b \rightarrow c1] ⟧.$$

We shall show that $(B \wedge C1)^+ \circ (B \wedge C2)^+ = (B \wedge C1)^+$. Denoting, as before, the left-hand side of this statement by P1 while the right-hand side by P2, we shall set up systems of equations using the new formalism.

Then, the value $\boldsymbol{\nu}$ should be modified. Taking into account new definition of tests, the value $\boldsymbol{\nu}$ for expression of the form $(B1 \wedge C1 + B2 \wedge C2 + . . .+ Bn \wedge Cn)^+$ may be defined as $\boldsymbol{\nu} = \beta \wedge \tau$, where $\beta = \rceil B1 * \rceil B2 * . . . * \rceil Bn$.

Thus, the systems of equations may be written in the following form:

$P11 = P1 = (B \wedge C1) \circ (B \wedge C1)^+ \circ (B \wedge C2)^+ + (\rceil B \wedge \tau) \circ$
$\circ (B \wedge C2)^+ =$
$= B \wedge C1 \circ (B \wedge C1)^+ \circ (B \wedge C2)^+ + (\rceil B \wedge \tau) \circ (B \wedge C2) \circ$
$\circ (B \wedge C2)^+ + (\rceil B \wedge \tau) \circ (\rceil B \wedge \tau) =$
$= B \wedge P12 + \rceil B \wedge (B \wedge C2) \circ (B \wedge C2)^+ + \rceil B \wedge \tau =$
$= B \wedge P12 + (\rceil B * B) \wedge C2 \circ (B \wedge C2)^+ + \rceil B \wedge \tau =$
$= B \wedge P12 + F \wedge C2 \circ (B \wedge C2)^+ + \rceil B \wedge P13 =$
$= B \wedge P12 + Ø + \rceil B \wedge P13 = B \wedge P12 + \rceil B \wedge P13$,
$P12 = C1 \circ P11$,
$P13 = \tau$,
$P21 = P2 = (B \wedge C1) \circ (B \wedge C1) + \rceil B \wedge \tau =$
$= B \wedge C1 \circ (B \wedge C1)^+ + \rceil B \wedge \tau =$
$= B \wedge P22 + \rceil B \wedge P23$,
$P22 = C1 \circ P21$,
$P23 = \tau$.

The following two systems of recursive equations may be written:

$P11 = B \wedge P12 + \rceil B \wedge P13$,
$P12 = C1 \circ P11$,
$P13 = \tau$,

and

$P21 = B \wedge P22 + \rceil B \wedge P23$,
$P22 = C1 \circ P21$,
$P23 = \tau$.

The resulting systems of equations coincide with the accuracy of variable denoting, which implies the equivalence of expressions P1 and P2.

We shall consider another example of applying the proposed method to proving another statement. Suppose we need to prove the following statement:

$\mathbb{C}$ ⟦while E do C1 od; while E do C2; while E do C2 od od ⟧ =
$\mathbb{C}$ ⟦if E then C1; while E do C1 od; while E do C2 od else e fi⟧.

In our formalism, the above statement may be written in the following form:

$*[b \rightarrow c1];*[b \rightarrow c2;*[b \rightarrow c2]] = (b \rightarrow [c1;*[b \rightarrow c1];*$
$*[b \rightarrow c2]) \square \ \bar{b} \rightarrow \text{skip}]$.

Let: $B = E[b]\rho$, $\rceil B = E[\bar{b}]\rho$, $C1 = C[c1]\rho$, $C2 = C[c2]\rho$.

$P1 = C[LHS]\rho = P11 = (B \wedge C1)^+ \circ (B \wedge (C2 \circ$
$\circ (B \wedge C2)^+))^+ = ((B \wedge C1) \circ (B \wedge C1)^+ + \rceil B \wedge \tau) \circ (B \wedge$
$\wedge (C2 \circ (B \wedge C2)^+))^+ = B \wedge P12 + Ø + \rceil B \wedge P13 = B \wedge P12 +$
$+ \rceil B \wedge P13$,
$P12 = C1 \circ (B \wedge C1)^+ \circ (B \wedge (C2 \circ (B \wedge C2)^+))^+ =$
$= C1 \circ P11$,
$P13 = \tau$.

Thus, the following may be written:

$P11 = B \wedge P12 + \rceil B \wedge P13$,
$P12 = C1 \circ P11$,
$P13 = \tau$,

$P2 = C[RHS]\rho = P21 = B \wedge (C1 \circ (B \wedge C1)^{+} \circ$
$\circ (B \wedge C2)^{+}) + \urcorner B \wedge \tau = B \wedge P22 + \urcorner B \wedge P23,$
$P22 = C1 \circ (B \wedge C1)^{+} \circ (B \wedge C2)^{+} = C1 \circ P11,$
$P23 = \tau.$

Thus, the following may be written:

$P21 = B \wedge P22 + \urcorner B \wedge P23,$
$P22 = C1 \circ P21,$
$P23 = \tau.$

The resulting systems of equations coincide with the accuracy of variable denoting, which implies the equivalence of expressions P1 and P2.

CONCLUSIONS

In the paper, the algebraic method for proving the equivalence of sequential program schemes was developed. To solve the problem, the algebraic model of semantic domain representing the set of all computation sequences of a program was proposed. The axiomatic system describing properties of operations over this semantic domain was given, as well as the presentability of axiomatic values for programs (program schemes) in the form of finite systems of recursive equations was proved. The algorithm for proving the equivalence of resulting systems of equations was proposed. Demonstration examples of proving the equivalence (non-equivalence) of program schemes applying the proposed method were given to illustrate its efficiency. This approach is not inferior to the well-known de Bakker–Scott fixed-point induction method in its capabilities, but as seen from the given examples, it simplifies proving the equivalence of program schemes significantly.

REFERENCES

1. Korablin Yu.P., Kulikova N.L., Shipov A.A. Programs as least fixed points: theoretical and applied aspects. *Cloud of Science*. 2020;7(3):535−550 (in Russ.).
2. Karpov Yu.G. *Model checking. Verifikatsiya parallel'nykh i raspredelennykh programmnykh system* (*Model checking. Verification of parallel and distributed software systems*). St. Petersburg: BHV-Petersburg; 2010. 610 p. (in Russ.). ISBN 978-5-9775-0404-1
3. Korablin Yu.P., Kuchugurov I.V. Process semantics of distributed programming language. *Programmnye produkty i sistemy = Software & Systems*. 2011;4(96):57−64 (in Russ.).
4. Korablin Yu.P. *Semanticheskie metody analiza programm* (*Semantic methods of program analysis*). Moscow: MEI; 2019. 68 p. (in Russ.). ISBN 978-5-7046-2173-7
5. Floyd R.W. Assigning meanings to programs. In: Colburn T.R., Fetzer J.H., Rankin T.L. (Eds.). *Program Verification. Studies in Cognitive Systems*. Dordrecht: Springer; 1993. V. 14. P. 65−81. https://doi.org/10.1007/978-94-011-1793-7_4
6. Hoare C.A.R., Wirth N. An axiomatic definition of the programming language PASCAL. *Acta Informatica.* 1973;2(4):335−355. https://doi.org/10.1007/BF00289504
7. Stoy J.E. *Denotational semantics: The Scott-Strachey approach to programming language theory.* Cambridge: The MIT Press Series in Computer Science; 1985. 414 p.
8. Alagić S., Arbib M. *The design of well-structured and correct programs*. NY: Springer; 1978. 292 p.
   [Alagich S., Arbib M. *Proektirovanie korrektnykh strukturirovannykh programm* (*Designing correct structured programs*): transl. from Eng. Moscow: Radio i svyaz'; 1984. 264 p. (in Russ.).]

СПИСОК ЛИТЕРАТУРЫ

1. Кораблин Ю.П., Куликова Н.Л., Шипов А.А. Программы как минимальные фиксированные точки: теоретические и прикладные аспекты. *Cloud of Science*. 2020;7(3):535−550.
2. Карпов Ю.Г. *Model checking. Верификация параллельных и распределенных программных систем.* СПб.: БХВ-Петербург; 2010. 610 с. ISBN 978-5-9775-0404-1
3. Кораблин Ю.П., Кучугуров И.В. Процессная семантика языков распределенного программирования. *Программные продукты и системы*. 2011;4(96):57−64.
4. Кораблин Ю.П. *Семантические методы анализа программ.* М.: Изд-во МЭИ; 2019. 68 с. ISBN 978-5-7046-2173-7
5. Floyd R.W. Assigning Meanings to Programs. In: Colburn T.R., Fetzer J.H., Rankin T.L. (Eds.). *Program Verification. Studies in Cognitive Systems*. Dordrecht: Springer; 1993. V. 14. P. 65−81. https://doi.org/10.1007/978-94-011-1793-7_4
6. Hoare C.A.R., Wirth N. An axiomatic definition of the programming language PASCAL. *Acta Informatica.* 1973;2(4):335−355. https://doi.org/10.1007/BF00289504
7. Stoy J.E. *Denotational semantics: The Scott-Strachey approach to programming language theory.* Cambridge: The MIT Press Series in Computer Science; 1985. 414 p.
8. Алагич С., Арбиб М. *Проектирование корректных структурированных программ*: пер. с англ. М.: Радио и связь; 1984. 264 с.
9. de Bakker J. *Mathematical Theory of Program Correctness*. Prentice Hall International; 1980. 505 p.
10. Захаров В.А. Моделирование и анализ поведения последовательных реагирующих программ. *Труды Института системного программирования РАН*. 2015:27(2):221−250. https://doi.org/10.15514/ISPRAS-2015-27(2)-13

9. de Bakker J. *Mathematical theory of program correctness*. Prentice Hall International; 1980. 505 p.
10. Zakharov V.A. Modeling and analysis of the behavior of successive reactive programs. *Trudy Instituta sistemnogo programmirovaniya RAN = Proceedings of the Institute for System Programming of the RAS*. 2015;27(2):221–250 (in Russ.). https://doi.org/10.15514/ISPRAS-2015-27(2)-13
11. Zakharov V.A., Podymov V.V. On the application of equivalencecheckingalgorithmsforprogramminimization. *Trudy Instituta sistemnogo programmirovaniya RAN = Proceedings of the Institute for System Programming of the RAS*. 2015;27(4):145–174 (in Russ.). https://doi.org/10.15514/ISPRAS-2015-27(4)-8
12. Zakharov V.A., Zhailauova S.R. On the minimization problem for sequential programs. *Modelirovanie i analiz informatsionnykh system = Modeling and Analysis of Information Systems*. 2017;24(4):415–433 (in Russ.). https://doi.org/10.18255/1818-1015-2017-4-415-433
13. Molchanov A.E. A Solution to the equivalent transformation problem in a class of primitive program schemes. *Trudy Instituta sistemnogo programmirovaniya RAN = Proceedings of the Institute for System Programming of the RAS*. 2015;27(2):173–188 (in Russ.). https://doi.org/10.15514/ISPRAS-2015-27(2)-11
14. Korablin Yu.P., Shipov A.A. Systems model verification based on equational characteristics of CTL formulas. *Programmnye produkty i sistemy = Software & Systems*. 2019;32(4):547–555 (in Russ.). http://dx.doi.org/10.15827/0236-235X.128.547-555

11. Захаров В.А., Подымов В.В. Применение алгоритмов проверки эквивалентности для оптимизации программ. *Труды Института системного программирования РАН*. 2015;27(4):145–174. https://doi.org/10.15514/ISPRAS-2015-27(4)-8
12. Захаров В.А., Жайлауова Ш.Р. О задаче минимизации последовательных программ. *Моделирование и анализ информационных сист*ем. 2017;24(7):415–433. https://doi.org/10.18255/1818-1015-2017-4-415-433
13. Молчанов А.Э. Разрешимость проблемы эквивалентных преобразований в классе примитивных схем программ. *Труды Института системного программирования РАН*. 2015;27(2):173–188. https://doi.org/10.15514/ISPRAS-2015-27(2)-11
14. Кораблин Ю.П., Шипов А.А. Верификация моделей систем на базе эквациональной характеристики формул CTL. *Программные продукты и системы*. 2019;32(4):547–555. http://dx.doi.org/10.15827/0236-235X.128.547-555

About the author

Yuri P. Korablin, Dr. Sci (Eng.), Professor, Professor, Department of Applied Mathematics and Artificial Intelligence, Institute of Information and Computational Technologies, the National Research University "MPEI" (14, Krasnokazarmennaya ul., Moscow, 111250 Russia). E-mail: KorablinYP@mpei.ru. Scopus Author ID 6603265252.

Об авторе

Кораблин Юрий Прокофьевич, д.т.н., профессор, профессор кафедры прикладной математики и искусственного интеллекта Института информационных и вычислительных технологий Национального исследовательского университета «МЭИ» (111250, Россия, Москва, Красноказарменная ул., д. 14). E-mail: KorablinYP@mpei.ru. Scopus Author ID 6603265252.

Translated by K. Nazarov.
The abstract was edited for English language and spelling by L. Daghorn, Awatera.