Russian Technological Journal.
2022;10(1):18–27

ISSN 2782-3210 (Print)
ISSN 2500-316X (Online)

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

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.

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.

ВВЕДЕНИЕ

Вопросы эквивалентности программ являются чрезвычайно важным аспектом теории и практики языков программирования, лежащим в основе таких проблем, как корректность программ [1–10], завершимость (незавершимость) программ, эквивалентных преобразований программ с целью их оптимизации [11–13] в том или ином виде. Очевидно, что для решения этих проблем необходима разработка соответствующих формальных методов задания семантики языков программирования. Одними из первых были разработаны пропозициональные методы, наиболее известными из которых являются метод индуктивных утверждений Флойда [5] и аксиоматический метод Хоара [6]. Использование этих методов позволяет осуществлять доказательство частичной корректности, завершимости (незавершимости) достаточно большого класса программ ограниченного размера. Денотационный подход [7] дает больше возможностей для решения проблемы эквивалентности программ, используя методы, базирующиеся на свойствах фиксированных точек, в частности, метод индукции фиксированных точек (де Баккера – Скотта) [4, 7]. В статье предложен алгебраический метод, задающий процессную семантику программ, которая сопоставляет программам в качестве семантических значений множество вычислительных последовательностей (путей выполнения) программы. Предложен алгоритм анализа эквивалентности схем программ, основанный на идее представления семантических значений в виде конечных систем рекурсивных уравнений с последующим анализом на эквивалентность (неэквивалентность) полученных систем рекурсивных уравнений, развиваемый автором в работах[1] [14]. Доказана возможность представления семантических значений в виде конечных систем рекурсивных уравнений. В отличие от метода, предложенного для доказательства эквивалентности схем программ[1], полученные в системах рекурсивные уравнения имеют более сложный вид и требуют более детального анализа. Показана эффективность данного метода для доказательства эквивалентности схем программ.

[1] Кораблин Ю.П. Семантические методы анализа распределенных систем: автореф. дис. докт. техн. наук. М.: МЭИ; 1994. 40 с. [Korablin Yu.P. Semantic methods of analysis of distributed systems. Dr. Thesis (Eng.). Moscow: MEI; 1994. 40 p. (in Russ.).]

1. ФОРМАЛЬНАЯ МОДЕЛЬ ЯЗЫКА ПРОГРАММИРОВАНИЯ

Определим синтаксис и семантику языка L, используемого нами в качестве модели языка программирования.

1.1. Синтаксис языка L

Алфавит языка состоит из:

- множества элементарных команд Com с типичным элементом a;
- множества булевских выражений Exp с типичным элементом b.

Типичные элементы множеств могут допускать индексацию. Кроме того, выделим еще три константы: skip – пустая команда; tt и ff – тождественно истинное и тождественно ложное булевские значения.

Множество команд Cmd с типичным элементом c определим следующим образом:

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

где gc – типичный элемент множества защищенных команд Gcom со следующим синтаксисом:

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

$$g ::= tt \mid ff \mid b.$$

Фигурные скобки используются для обозначения нулевого либо большего количества повторений конструкции, заключенной в скобки. Через g обозначается типичный элемент множества защит G.

Символом □ соединяются альтернативные компоненты программы, т.е. такие конструкции являются аналогом оператора ветвления switch в языках программирования.

Множество программ с типичным элементом pr определяется следующим образом:

$$pr ::= c \mid pr; c.$$

1.2. Алгебраическая семантика: математические основы и семантические равенства

Семантическая функция сопоставляет программе в языке L множество вычислительных последовательностей (ВП), по которым может проходить выполнение данной программы.

Для построения семантики программы мы используем принцип построения семантического значения всей программы на основе семантических значений компонентов этой программы.

Полагая, что любая программа может, в свою очередь, быть компонентом другой программы, мы всегда получаем в результате композиции семантических значений компонентов некоторой программы априорную семантику всей программы.

1.2.1. Семантические области

1. Множество значений элементарных команд ACom с типичным элементом A.
2. Множество значений булевских выражений Bexp с типичным элементом B.
3. Множество Const, содержащее константы τ (тождественное преобразование), T (тождественно истинное значение) и F (тождественно ложное значение).

Типичные элементы семантических областей могут иметь индексы.

Определим множество вычислительных последовательностей CPath с типичным элементом cp.

Предварительно определим два дополнительных множества: множество тестов Test с типичным элементом β и множество действий Action с типичным элементом α.

$\beta ::= T \mid F \mid B$,

$\alpha ::= \tau \mid A$,

$cp ::= \alpha \mid \beta \wedge cp \mid cp1 \circ cp2$.

Обозначим через cp* конечную ВП, а через cpω – бесконечную ВП. Введем множество $SP = \mathcal{P}(CPath)$ с типичным элементом sp, т.е. SP – это множество всех подмножеств множества CPath.

Определим на множестве SP операции теоретико-множественного объединения (sp1 + sp2), последовательной композиции (sp1 ∘ sp2) и минимальной фиксированной точки (sp^+).

Прежде чем определить значение sp^+, введем два вспомогательных определения.

Определение 1. ВП cp называется пустой и обозначается через ε, если:

1) $cp \equiv \tau$;

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

Бесконечную последовательность вида εω обозначим через LOOP (зацикливание).

Определение 2. $\varepsilon \in sp$, если:

$sp \equiv \varepsilon$;

$sp = sp1 + sp2$ и ε принадлежит хотя бы одному из множеств sp1 или sp2;

$sp = sp1 \circ sp2$ и ε принадлежит и sp1, и sp2 одновременно.

Определим теперь sp^+ как минимальную фиксированную точку оператора F(sp), т.е. $sp+ = \mu F(sp)$, где F(sp) имеет вид:

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

1.2.2. Семантика языка L

Семантическая функция определяется следующим образом.

$\mathbb{C}[\![skip]\!] = \tau$,

$\mathbb{C}[\![a]\!] = A$,

$\mathbb{C}[\![tt \rightarrow pr]\!] = \mathbb{E}[\![tt]\!] \wedge \mathbb{C}[\![pr]\!]$,

$\mathbb{C}[\![ff \rightarrow pr]\!] = \mathbb{E}[\![ff]\!] \wedge \mathbb{C}[\![pr]\!]$,

$\mathbb{C}[\![b \rightarrow pr]\!] = \mathbb{E}[\![b]\!] \wedge \mathbb{C}[\![pr]\!]$,

где функция $\mathbb{E}$: Exp → TEST определяет семантические значения выражений.

$\mathbb{C}[\![*\{gc\}]\!] = (\mathbb{C}[\![gc]\!])^+$,

$\mathbb{C}[\![gc1 \square gc2]\!] = \mathbb{C}[\![gc1]\!] + \mathbb{C}[\![gc2]\!]$,

$\mathbb{C}[\![pr1; pr2]\!] = \mathbb{C}[\![pr1]\!] \circ \mathbb{C}[\![pr2]\!]$.

Определим теперь семантическую функцию для выражений:

$\mathbb{E}$: Exp → TEST,

$\mathbb{E}[\![tt]\!] = T$,

$\mathbb{E}[\![ff]\!] = F$,

$\mathbb{E}[\![b]\!] = B$.

1.3. Аксиоматизация свойств семантической области

Свойства операций над элементами семантической области $SP\sigma = \mathcal{P}(CPath\sigma)$ опишем системой аксиом (схем аксиом) и правил вывода. Система аксиом задается как набор блоков, каждый из которых описывает определенные свойства семантических объектов.

Обозначим через D′ множество ACTION ∪ TEST с типичным элементом d′. Для обозначения элементов семантической области будем использовать метапеременные X, Y, Z.

Аксиомы, задающие базисные свойства операций «∘», «^» и «+»:

(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) $\emptyset \circ X = \emptyset$,

(A10) $X + \emptyset = X$,

(A11) $(\beta \wedge X) \circ Y = \beta \wedge (X \circ Y)$,

(A12) $F \wedge X = \emptyset$,

(A13) $\emptyset \wedge X = \emptyset$.

2. ЭКВАЦИОНАЛЬНАЯ ХАРАКТЕРИЗАЦИЯ АПРИОРНЫХ СЕМАНТИЧЕСКИХ ЗНАЧЕНИЙ

В данном разделе показана возможность представления семантических значений программ в виде конечных систем рекурсивных уравнений.

Определение 3. Пусть $\alpha \in ACT = ACTION \setminus \{\tau, Ø\}$ и ($\beta \in TES = TEST \setminus \{F, Ø\}$ и $PREF = ACT \cup TES$. Частичные функции prefix: $SP \to PREF$ и suffix: $SP \to SP$ определяются таблицей 1:

Таблица 1. Определение префиксов и суффиксов вычислительных последовательностей (sp)

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

Заметим, что для выражения $sp \equiv F \wedge X$ понятие префикса и суффикса не определяются, поскольку $F \wedge X \equiv Ø$ (аксиома A12).

Определение 4. Множество ВП $P \in SP\sigma$ эквационально характеризуемо, если имеется конечное множество $P1, P2, ..., Pn \in SP$, таких, что $P \equiv P1$ и для любого i ($1 \le i \le n$):

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

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

$\forall i\, \delta(Pi) \subset \mathcal{P}(ACT' \cup \{LOOP\})$, где $ACT' = ACTION \setminus \{\tau\}$,

$\forall i\ \forall j\ \exists l \in N$ такое, что $Pij = Pl$,

$\forall i\ \forall k\ \exists r \in N$ такое, что $Pik = Pr$.

Теорема 1. Каждое множество SP, сопоставляемое программе pr в качестве семантического значения, может быть эквационально характеризуемо с помощью конечной системы уравнений вида (*).

Доказательство. Доказательство проведем методом индукции по структуре P.

Базис. Для $p \equiv d'$, где $d' \in D'$, эквациональная характеризация вытекает тривиально.

Индуктивный шаг. Пусть $P1 = SP$ и $P2 = SP$ будут эквационально характеризуемы. Тогда необходимо доказать, что $P1 \circ P2$, $\beta \wedge P1$, $P1 + P2$ и $P1^+$ эквационально характеризуемы.

Приведем доказательство для выражения $P = P1 \circ P2$.

По предположению индукции существуют множества $P11, P12, ..., P1n$ и $P21, P22, ..., P2m$, такие что $P1 \equiv P11$ и $P2 \equiv P21$, и

$$P1i = \sum_{j \in N} \alpha_{ij} \circ P1_{ij} + \sum_{k \in N} \beta_{ik} \wedge P1_{ik} + \delta(P1i),\ i = \overline{1,n}, \quad (**)$$

и

$$P2i = \sum_{r \in N} \alpha_{ir} \circ P2_{ir} + \sum_{p \in N} \beta_{ip} \wedge P2_{ip} + \delta(P2i),\ i = \overline{1,m}, \quad (***)$$

Обозначим

$$\eta(u, v1, ..., vr) = P1_u \circ P2 + P2_{v_i} + ... + P2_{v_r}, \quad (u = \overline{0,n},\ r \ge 0,\ 1 \le vi \le m,\ i = \overline{1,r}). \qquad (1)$$

Будем писать $P10 \circ P2 + P2_{v_i} + ... + P2_{v_r}$ вместо $P2_{v_i} + ... + P2_{v_r}$.

Количество выражений (1) конечно. По индуктивному предположению имеем:

$$\eta(u, v1, ..., vr) = \left(\sum_{j \in N} \alpha_{uj} \circ P1_{uj} + \sum_{k \in N} \beta_{uk} \wedge P1_{uk} + \delta(P1u) \right) \circ P2 + \sum_{j \in N} \alpha_{v_1 j} \circ P2_{v_1 j} + \sum_{k \in N} \beta_{v_1 k} \wedge P2_{v_1 k} + \delta(P2v_1) + ... + \sum_{j \in N} \alpha_{v_r j} \circ P2_{v_r j} + \sum_{k \in N} \beta_{v_r k} \wedge P2_{v_r k} + \delta(P2v_r).$$

Применяя далее аксиомы A1, A2, A3 и A5, получим:

$$\eta(u, v1, ..., vr) = \sum_{j \in N} \alpha_{uj} \circ P1_{uj} \circ P2 + \sum_{k \in N} \beta_{uk} \wedge (P1_{uk} \circ P2) + \sum_{j \in N} \alpha_{v_1 j} \circ P2_{v_1 j} + \sum_{k \in N} \beta_{v_1 k} \wedge P2_{v_1 k} + ... + \sum_{j \in N} \alpha_{v_r j} \circ P2_{v_r j} + \sum_{k \in N} \beta_{v_r k} \wedge P2_{v_r k} + \delta(P1u) \circ P2 + \delta(P2v_1) + ... + \delta(P2v_r).$$

Если $\delta(P1u) = \sum_{j \in N} \delta_{ui}$, где $\delta ui \in D'$, то применим еще раз аксиому A4, заменяя в $\eta(u, v1, ..., vr)$ слагаемое $\delta(P1u) \circ P2$ выражением $\sum_{j \in N} \delta_{ui} \circ P2$, причем, если $\delta_{ui} = \tau$, то P2 заменяем его представлением для P21 из (***).

Применяя далее аксиомы A1, A2, A3 и A5, получим:

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

где все выражения P_{uj}, P_{uk}, P_{qj} и P_{qk} входят в (1). Поскольку $\eta(1,1) \equiv P1 \circ P2$, получаем, что $P1 \circ P2$ эквационально характеризуемо.

$P = \beta \wedge P1$. Тривиальный случай. В этом случае к конечной системе уравнений для $P1 = P11$ добавляется лишь одно уравнение: $P = P1 = \beta \wedge P11$, откуда следует эквациональная характеризуемость выражения $\beta \wedge P1$.

Доказательство эквациональной характеризации выражения $P = P1 + P2$ осуществляется аналогичным способом, где в качестве конечного множества используется множество

$$\xi(u,v) \equiv P1u + P2v,\ u = \overline{0,n};\ v = \overline{0,m}. \qquad (2)$$

Для доказательства эквациональной характеризации выражения $P = P1^+$ используется множество

$$\zeta(u1,...,ur) \equiv (P1u_1 + ... + P1u_r) \circ P1^+,$$
$$r \geq 0,\ 1 \leq ui \leq n,\ i = \overline{1,r}. \qquad (3)$$

Количество выражений (3) конечно.

Таким образом, в качестве семантического значения программы задается множество путей выполнения программы, т.е. процессная семантика, позволяющая исследовать многие свойства программ. В частности, как будет показано ниже, этот подход позволяет исследовать вопросы эквивалентности схем программ в сильном смысле.

3. СРАВНЕНИЕ СХЕМ ПРОГРАММ В ЯЗЫКЕ L

Определим метод сравнения систем рекурсивных уравнений вида (*), что позволит нам получить формальный метод для сравнения схем программ в языке L.

Существенным фактором рассматриваемого метода является уникальность префиксов любого рекурсивного уравнения, что достигается посредством применения аксиомы вида $X \circ (Y + Z) = X \circ Y + X \circ Z$. В результате применения этой аксиомы любое рекурсивное уравнение приводится к виду, где все α_{ij} попарно различны.

Таким образом, процесс доказательства эквивалентности (либо не эквивалентности) двух систем рекурсивных уравнений на каждом шаге сводится к доказательству эквивалентности пар выражений, которым соответствует одинаковые префиксы в рассматриваемых рекурсивных уравнениях.

При этом процесс сравнения завершается, когда перестанут появляться новые пары выражений, либо на некотором шаге будет получено несовпадение множеств префиксов или выражений δ для некоторой пары выражений.

Первый случай завершения процесса сравнения означает эквивалентность двух систем рекурсивных уравнений, а второй случай означает их несравнимость, а, следовательно, неэквивалентность двух систем рекурсивных уравнений.

Пусть даны две системы рекурсивных уравнений вида (*), которые мы хотим исследовать на эквивалентность. В этих системах заданы рекурсивные уравнения для выражений P1, P2, …, Pn и P1′, P2′, ..., Pm′, соответственно. Обозначим через P множество выражений {P1, P2, ..., Pn}, через P′ – {P1′, P2′, ..., Pm′}, через $\mathcal{P}$(P) и $\mathcal{P}$(P′) – множества всех подмножеств P и P′, соответственно.

Рассмотрим уравнения для P1 и P1′:

Возможны следующие ситуации:

а) $\delta(P1) = \delta(P1')$ и $\forall j \in N\ \exists k \in N$ такое, что $\alpha 1j \equiv \alpha 1k'$ и наоборот, а также $\forall k \in N\ \exists p \in N$ такое, что $\beta 1k \equiv \beta 1p'$ и наоборот. В этом случае продолжаем процесс выписывания уравнений для всех пар (P_{1j}, P_{1k}), имеющих одинаковые префиксы типа α, а также для всех пар (P_{1k}, P_{1p}), имеющих одинаковые префиксы типа β.

б) не выполнено хотя бы одно из условий пункта а). Это случай несравнения множества префиксов, либо несравнения свободных членов уравнений ($\delta(P1) \neq \delta(P1')$), откуда следует, что множества вычислительных последовательностей, задаваемых системами рекурсивных уравнений, несравнимы.

Представленную выше процедуру выписывания уравнений продолжаем для получаемых пар до получения одного из результатов:

- построены две системы рекурсивных уравнений, эквивалентные с точностью до обозначений (на каждом шаге построения систем только ситуация а) имела место). В этом случае множества вычислительных последовательностей, задаваемых системами рекурсивных уравнений, эквивалентны, а, следовательно, схемы программ, которым сопоставлены эти выражения, эквивалентны;
- не выполнено условие а). В этом случае множества вычислительных последовательностей, задаваемых системами рекурсивных уравнений, не эквивалентны, а, следовательно, схемы программ, которым сопоставлены эти выражения, не эквивалентны;

Проанализируем эффективность предложенного метода доказательства эквивалентности схем программ. В частности, рассмотрим вначале пример доказательства с использованием широко используемого метода индукции фиксированной точки [4] справедливости следующего утверждения:

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

Решение. Введем следующие обозначения: $\mathbb{E}$ ⟦E⟧=ω; $\mathbb{C}$ ⟦C1⟧ = γ1; $\mathbb{C}$ ⟦C2⟧ = γ2.

ℂ ⟦while E do C1 od;⟧ = fix(λγ.λσ.ωσ → γ·γ1σ,σ) = = fix H1 = γ1′,
где запись fix H1 обозначает взятие минимальной фиксированной точки оператора H1 = λγ.λσ.ωσ → → γ·γ1σ,σ.

ℂ ⟦while E do C2 od;⟧ = fix(λγ.λσ.ωσ → γ·γ1σ,σ) = = fix H2 = γ2′,
где H2 = λγ.λσ.ωσ → γ·γ1σ,σ.

Левая часть (л.ч.) уравнения задается следующим образом:

л.ч. = ℂ ⟦while E do C2 od;⟧ · ℂ ⟦while E do C1 od;⟧ = = γ2′· γ1′.

Правая часть (п.ч.) уравнения, соответственно, имеет вид:

п.ч. = γ1′.

Таким образом, требуется доказать, что: γ2′· γ1′ = γ1′.

Доказательство проведем методом индукции фиксированной точки

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

1. $q(\perp) \equiv \gamma 2' \cdot \perp = \perp$

2. Пусть $q(x) \equiv \gamma 2' \cdot x = x$ – справедливо.

Покажем, что $q(H1(x)) \equiv \gamma 2' \cdot H1(x) = H1(x)$ справедливо.

л.ч. = $\gamma 2' \cdot H1(x) = \gamma 2' \cdot \lambda\sigma.\omega\sigma \to x \cdot \gamma 1\sigma,\sigma$ = (левая факторизация)

= $\lambda\sigma.\omega\sigma \to \gamma 2' \cdot x \cdot \gamma 1\sigma, \gamma 2'\sigma$ = (по предположению о справедливости $q(x)$)

= $\lambda\sigma.\omega\sigma \to x \cdot \gamma 1\sigma, \gamma 2'\sigma$ = (по свойству фиксированной точки)

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

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

= $\lambda\sigma.\omega\sigma \to x \cdot \gamma 1\sigma, (\lambda\sigma.\omega\sigma \to \gamma 2' \cdot \gamma 2\sigma,\sigma)$ = (по свойству условного оператора)

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

п.ч. = $H1(x) = \lambda\sigma.\omega\sigma \to x \cdot \gamma 1\sigma, \sigma$.

Из п. 1 и п. 2 вытекает, что $q(\text{fix } H1) = q(\gamma 1')$ – справедливо, а следовательно

$\gamma 2' \cdot \gamma 1' = \gamma 1'$, что и требовалось доказать.

В предложенной нами нотации это утверждение может быть записано следующим образом:

*[b → c1]; *[b → c2] = *[b → c1].

Используя предложенный выше алгоритм, докажем справедливость утверждения

ℂ ⟦ *[b → c1]; *[b → c2] ⟧ = ℂ ⟦ *[b → c1] ⟧.

Обозначим ℂ ⟦ *[b → c1]; *[b → c2] ⟧ через P1, а ℂ ⟦ *[b → c1] ⟧ через P2.

$P1 = (B \wedge C1)^+ \circ (B \wedge C2)^+$,

$P2 = (B \wedge C1)^+$.

Представим теперь P1 и P2 системами рекурсивных уравнений в соответствии с заданным выше алгоритмом сравнения схем программ. Пусть P11 ≡ P1 и P21 ≡ P2. Тогда имеем:

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

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

Нетрудно заметить, что множество P12 содержит вычислительные последовательности, начинающиеся с C2, тогда как множество P22 не содержит вычислительных последовательностей, начинающихся с C2, откуда следует, что P12 ≠ P22.

На первый взгляд данный результат противоречит тому, что было получено выше. Однако суть проблемы заключается в том, что при получении этого результата мы предполагали отсутствие побочного эффекта при вычислении выражений (условий выполнения защищенных команд). Из этого предположения непосредственно следовало, что повторное вычисление одного и того же выражения всегда дает один и тот же результат. В нашем же случае повторное вычисление одного и того же выражения может привести, в общем случае, к получению различных результатов. Проиллюстрируем этот факт следующим примером. Пусть дана программа, допускающая побочный эффект, т.е. изменение значений переменных при вычислении выражений,

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

При первоначальном вычислении выражения x: = 2 × x = 4 будет получено ложное значение, а следовательно, будет осуществлен выход из цикла *[x: = 2 × x = 4 → c1] и переход к вычислению следующей команды. При этом значение переменной x перед выполнением будет равно 2. Повторное вычисление выражения x: = 2 × x = 4 даст истинное значение. Очевидно, что наличие побочного эффекта является, как правило, нежелательной ситуацией. Поэтому целесообразным является построение формализма, позволяющего осуществлять анализ программ, не допускающих побочного эффекта. Для этой цели нам понадобится переопределить ряд понятий, введенных ранее, и добавить аксиомы, характеризующие их свойства.

Расширим вначале множество тестов, добавив в него отрицание и сложные тесты.

β ::= . . . | ⅂β | β1 ∗ β 2 , где под многоточием понимается ранее определенное множество тестов.

Новое множество тестов характеризуется следующим блоком аксиом:

(G1) β1 ∗ β2 = β2 ∗ β1,
(G2) β ∗ ⅂β = F,
(G3) F ∗ β = F,
(G4) Ø ∗ β = Ø,
(G5) T ∗ β = β,
(G6) β ∗ β = β,
(G7) β1 ∗ (β2∗ β3) = (β1∗ β2) ∗ β3,
(G8) β1 ^ (β2 ^ C) = (β1 ∗ β2) ^ C.

Расширение множества тестов ведет к необходимости модифицировать ранее определенные понятия префикса и суффикса.

Определение 5. Частичные функции prefix: SP → PREF, и suffix: SP → SP определяются следующей таблицей 2, являющейся расширением таблицы 1:

Таблица 2. Определение префиксов и суффиксов вычислительных последовательностей (sp)

SP	prefix (SP)	suffix (SP)
α ∘ X	α	X
τ ∘ X	prefix (X)	suffix (X)
β1 ∘ X1, где X1 ≠ β2 ^ X2, и X1 ≠ τ ∘ X, и X1 ≠ X2$^+$	β1	X1
β1 ^ (β2 ^ X)	prefix ((β1 * β2) ^ ^ X)	Suffix ((β1 * β2) ^ ^ X)
β ^ (τ ∘ X)	prefix (β ^ X)	suffix (β ^ X)

Отметим, что в том случае, когда sp ≡ β1 ^ X1 и X1≡ X2$^+$, необходимо предварительно воспользоваться свойством минимальной фиксированной точки до появления одной из конструкций, для которой в вышеприведенной таблице определены префикс и суффикс.

Докажем теперь ранее приведенное утверждение
ℂ ⟦ *[b → c1]; *[b → c2] ⟧ = ℂ ⟦ *[b → c1] ⟧.

Покажем, что (B ^ C1)$^+$ ∘ (B ^ C2)$^+$ = (B ^ C1)$^+$. Обозначая, как и ранее, левую часть этого утверждения через P1, а правую через P2, построим системы уравнений, используя новый формализм.

Далее необходимо модифицировать значение **ν**. С учетом нового определения тестов значение **ν** для выражения вида (B1 ^ C1 + B2 ^ C2 +. . .+ Bn ^ Cn)$^+$ определяется как **ν** = β ^τ, где β = ⌉B1 ∗ ⌉B2 ∗. . . ∗ ⌉Bn.

Таким образом, системы уравнений принимают следующий вид:

P11 = P1= (B ^ C1) ∘ (B ^ C1)$^+$ ∘ (B ^ C2)$^+$ + (⌉B ^ τ) ∘ ∘ (B ^ C2)$^+$ =
= B ^ C1 ∘ (B ^ C1)$^+$ ∘ (B ^ C2)$^+$ + (⌉B ^ τ) ∘ (B ^ C2) ∘ ∘ (B ^ C2)$^+$ + (⌉B ^ τ) ∘ (⌉B ^ τ) =
= B ^ P12 + ⌉B ^ (B ^ C2) ∘ (B ^ C2)$^+$ + ⌉B ^ τ =
= B ^ P12 + (⌉B ∗ B) ^ C2 ∘ (B ^ C2)$^+$ + ⌉B ^ τ =
= B ^ P12 + F ^ C2 ∘ (B ^ C2)$^+$ + ⌉B ^ P13 =
= B ^ P12 + Ø + ⌉B ^ P13= B ^ P12 + ⌉B ^ P13,
P12 = C1 ∘ P11,
P13 = τ,
P21 = P2 = (B ^ C1) ∘ (B ^ C1) + ⌉B ^ τ =
= B ^ C1 ∘ (B ^ C1)$^+$ + ⌉B ^ τ =
= B ^ P22 + ⌉B ^ P23,
P22 = C1 ∘ P21,
P23 = τ.

Мы получили две системы рекурсивных уравнений:

P11 = B ^ P12 + ⌉B ^ P13,
P12 = C1 ∘ P11,
P13 = τ

и

P21 = B ^ P22 + ⌉B ^ P23,
P22 = C1 ∘ P21,
P23 = τ.

Полученные системы уравнений совпадают с точностью до обозначения переменных, откуда вытекает эквивалентность выражений P1 и P2.

Рассмотрим еще один пример применения предложенного метода для доказательства еще одного утверждения. Пусть требуется доказать справедливость следующего утверждения:

ℂ ⟦while E do C1 od; while E do C2; while E do C2 od od ⟧ =

ℂ ⟦if E then C1; while E do C1 od; while E do C2 od else e fi⟧.

В нашем формализме приведенное утверждение примет вид:

[b → c1];[b → c2;*[b → c2]] = (b → [c1;*[b → c1]; *[b → c2]) □ $\bar{b}$ → skip].

Пусть: B = E[b]ρ, ⌉B = E[$\bar{b}$]ρ, C1 = C[c1]ρ, C2 = C[c2]ρ.

P1 = C[л.ч.]ρ = P11 = (B ^ C1)$^+$ ∘ (B ^ (C2 ∘ (B ^ C2)$^+$))$^+$ = = ((B ^ C1) ∘ (B ^ C1)$^+$ + ⌉B ^ τ) ∘ (B ^ (C2 ∘ (B ^ C2)$^+$))$^+$ = = B ^ P12 + Ø + ⌉B ^ P13 = B ^ P12 + ⌉B ^ P13,

P12 = C1 ∘ (B ^ C1)$^+$ ∘ (B ^ (C2 ∘ (B ^ C2)$^+$))$^+$ = = C1 ∘ P11,

P13 = τ.

Таким образом, имеем:

P11 = B ^ P12 + ⌉B ^ P13,
P12 = C1 ∘ P11,
P13 = τ.

P2 = C[п.ч.]ρ = P21 = B ^ (C1 ∘ (B ^ C1)$^+$ ∘ ∘ (B ^ C2)$^+$) + ⌉B ^ τ = B ^ P22 + ⌉B ^ P23,

P22 = C1 ∘ (B ^ C1)$^+$ ∘ (B ^ C2)$^+$ = C1 ∘ P11,

P23 = τ.

Таким образом, имеем:

P21 = B ^ P22 + ⌉B ^ P23,
P22 = C1 ∘ P21,
P23 = τ.

Полученные системы уравнений совпадают с точностью до обозначения переменных, откуда вытекает эквивалентность выражений P1 и P2.

ЗАКЛЮЧЕНИЕ

В статье разработан алгебраический метод доказательства эквивалентности схем последовательных программ. Для решения поставленной задачи предложена алгебраическая модель семантической области, представляющей собой множество всех вычислительных последовательностей программы. Приведена система аксиом, описывающая свойства операций над этой семантической областью, и доказана представимость аксиоматических значений программ (схем программ) в виде конечных систем рекурсивных уравнений. Предложен алгоритм

доказательства эквивалентности полученных систем уравнений. Приведены демонстрационные примеры доказательства эквивалентности (неэквивалентности) схем программ предложенным методом, иллюстрирующие его эффективность. Данный подход не уступает по своим возможностям известному методу индукции фиксированной точки де Баккера – Скотта, но, как видно из приведенных примеров, значительно упрощает процесс доказательства эквивалентностей схем программ.

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
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 сис*тем. 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

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.).]
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 equivalence checking algorithms for program minimization. *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

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.

About the author

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