УДК 519.768

УРАВНЕНИЯ, ОПЕРАТОРЫ В ПРОСТРАНСТВАХ ПРОГРАММ И ОЦЕНКА СЛОЖНОСТИ

Казиев В.М., Казиева Б.В., Кодзоков А.Х., Нагоров А.Л.

ФГБОУ ВО «Кабардино-Балкарский государственный университет имени Х.М. Бербекова», Нальчик, e-mail: yka@kbsu.ru

В работе исследуются некоторые формально-математические операторы и операторные уравнения над пространством программ, которые полезны при построении и изучении формальных семейств программ, их верификации. Исследована также проблема релевантности и сложности программных систем. Исследованы операторы минимизации по числу операндов, операторов, времени программирования программы, показаны их функциональные свойства (с точки зрения математического функционального анализа). Исследованы существование и единственность неподвижной точки операторов, их линейность, однородность, монотонность, обратимость. Рассмотрен как реальный объем, так и граничный объем программы (без учета конкретного языка программирования). Системно исследуются проблемы сложности программ. Сложность бывает различного типа и форм, но в работе нас интересуют больше формы. В частности, программы, представимые графом (орграфом), деревом и вектор-функцией. В зависимости от глобальности используемых переменных проводится соответствующий анализ. В последнем случае длина определяется количеством глобальных переменных, а сложность программы оцениваем суммой его компонент. Аксиоматизирована сложность (согласно общей теории систем), приведен поучительный дидактический пример. Результаты помогут формализовать, построить оценочную систему программ, провести их верификацию, повысить релевантность, снизить сложность программ.

Ключевые слова: программа, пространства программ, метрики, меры, операторы, уравнения, релевантность, сложность, программирование

THE EQUATIONS, OPERATORS IN PROGRAMS SPACES AND COMPLEXITY ASSESSMENT

Kaziev V.M., Kazieva B.V., Kodzokov A.Kh., Nagorov A.L.

Federal Autonomous Educational Institution of Higher Education Kabardino-Balkaria State University named after Kh.M. Berbekov, Nalchik, e-mail: yka@kbsu.ru

In the work some formal mathematical operators and the operator equations in programs sets are probed. It is useful in case of creation and a research of the formal programs therefore it is researched their properties. Also a problem of complexity of programs and solutions of the program equations are probed. Operators of minimization on quantity of operands, operators, time of programming of the program is probed, their functional properties are studied (from the point of view of the mathematical functional analysis). Existence and uniqueness of a stationary point of operators, their linearity, homogeneity, a monotonicity are probed. We consider the actual volume and boundary volume of the program without communications with a certain programming language. Are systemically researched a problem of complexity of programs. The complexity happens different type, different form, but in article we are interested only in the form. In particular, programs which are representation the oriented graph, a tree and vectorial function. Depending on the variables used in the program the appropriate analysis is made. In the latter case length is determined by quantity of global variables, and we estimate complexity of the program as the amount of its components. The complexity of programs is set by axioms (according to the general systems theory), the effective didactic example is this. Results will help to formalize, construct the system of assessment of programs, to execute their check, to increase relevance, to reduce complexity of programs.

Keywords: program, spaces of programs, metrics, measures, operators, equations, relevance, complexity, programming

В работе исследуются некоторые операторы и операторные уравнения в пространстве программ, которые полезны при построении и изучении формальных семейств программ, верификации, оценки их сложности.

В [1, с. 14] рассмотрены некоторые операторные уравнения $A(x) = y$ в пространстве программ M над алфавитами X и Y, $M(P_i)$, $i = 1, 2,\ldots$ – множество программ которые реализуют одну задачу P_i, введены понятия сходимости, на базе которых рассмотрены операторы минимизации Ad, Af, At по числу операндов $d(x)$, операторов $f(x)$, времени программирования $t(x)$ программы x, соответственно, с критериями: $Ad(x) = y$, $\forall z \in P$: $d(y) \le d(z)$; $Af(x) = y$, $\forall z \in P$: $f(y) \le f(z)$; $At(x) = y$, $\forall z \in P$: $t(y) \le t(z)$. Но необходимая в теоретическом аспекте проблема исследования разрешимости операторных уравнений над пространствами программ не была исследована. Цель данной работы – исследовать разрешимость таких уравнений. Применяются методы функционального анализа, операторов сжатия и др. Рассматриваются и практические характеристики операторных уравнений, например оцениваются время, длина, сложность, устойчивость и другие. Методами системного анализа проведена аксиоматизация меры сложности модулей.

Время оцениваемо различными способами [2, с. 80], например по Холстеду:

$$t(y) = (Nf(y)\log_2 d(y)\log_2(d(y) + f(y))) / (2S),$$

$N = f(y)\log_2 f(y) + d(y)\log_2 d(y)$ – длина программы, S – число Страуда.

Продемонстрируем действие At на фрагменте программы решения $ax^2 + bx + c = 0$ (ищем действительные корни, описания, оптимальность не акцентируется):

```
read(a, b, c); d:=sqr(b)–4*a*c;
if (d<0) then write('нет корней')
        else begin
                x1:=(–b + sqrt(d) ) / (2*a); x2:=(–b – sqrt(d)) / (2*a);
                if (d=0) then write('кратный корень x=',x1)
    else write('корни x1=', x1,', x2=', x2)
            end;
```

Итак, количество операндов равно *14* (*a, b, c, d, 2, 0, 4, x, x1, x2* и четыре текста вывода); количество типов операторов – *16* («write», «read», «if – then – else», «:=», «/», «*», «–», «+», «<», «=», «sqr», «sqrt», «()», «;», «begin», «end»). При среднем значении *S=18* (обычно $3 < S < 25$) получаем $t \approx 972$ сек.

Заменяя все вхождения не входных-выходных переменных *d, x, x1, x2* на соответствующие выражения, получим

```
read(a, b, c);
if (sqr(b) – 4*a*c < 0) then write('корней нет') else begin
   if (sqr(b) – 4*a*c = 0) then write('кратный корень=',–b/(2*a))
   else if (sqr(b) – 4*a*c > 0) then
   write('x1=',(–b–sqrt(sqr(b)–4*a*c))/(2*a),'x2=',(–b+sqrt(sqr(b)–4*a*c))/(2*a));
end;
```

Для 16 операторов, 8 операндов, фиксируем новое время t = 538 сек. Время уменьшилось почти вдвое. Можно далее «вывести из программы» оператор sqr, вводя оператор D:= b*b–4*a*c; и уменьшить еще несколько время. Но оптимальный вариант программы, естественно, – по оптимальному алгоритму. В этой простой задаче можно предложить вариант, в котором используется основной блок вида:

```
if (sqr(b) – 4*a*c < 0) then write('нет корней')
  else if (sqr(b) – 4*a*c = 0)
   then write('кратный корень x=',–b/(2*a)) else begin
     x1:=–b/(2*a); x2:=x1*x1–c/a;
     if(x2>0) then begin
       x2:=sqrt(x2); x1:=x1–x2; x2:=x1+2*x2; write('x1=',x1,'x2=',x2)
end;
```

Затрачено меньше операций и операндов, чем в классическом алгоритме. И, хотя время уже уменьшилось лишь немного (до 455 сек), отметим, что «сэкономлены» наиболее длительные (с точки зрения микропрограммной реализации) операции – вычитание, вычисление корня, умножение.

Пространства программ, операторы над ними и их свойства

Рассмотрим множество программ M с несовпадающими входными множествами, например, решающих одну задачу различными методами. В пространстве X элементов $x_1, x_2, \ldots, x_n, \ldots$ вводится метрика

$$\rho(x, y) = \sum_{i,j=1}^{n} r_{ij},$$

n – количество модулей в x, y, r_{ij} – веса дуг орграфа $G = \langle \{(x_i, x_j)\}, \{r_{ij}\} \rangle$.

Разобьем M на M_1, M_2,…, $M_i \neq \emptyset$, $M_i \cap M_j = \emptyset$, чтобы в каждое подмножество попали программы, рассматриваемые (разрабатываемые) одинаковым методом. На множестве M^* оптимальных, минимизированных по вышеприведенным критериям программ на M_i, $i = 1, 2, \ldots$ выберем некоторую оптимальную по рассматриваемому параметру (длина, время, объем, полнота тестирования, сложность, устойчивость и др.) программу.

Рассмотрим вопрос о существовании и единственности неподвижной точки. Для оператора *Ad* (*Af*, *At*) она существует. Действительно, для любой программы *x* можно составить последовательность программ $X_{i+1} = A(X_i)$, $X_0 = X$, $i = 1, 2,\ldots$, тогда для любого *i* выполняется условие

$$d(x_i) \geq d(x_{i+1}) \left(f(x_i) \geq f(x_{i+1}), t(x_i) \geq t(x_{i+1})\right),$$

но функция $d(x)$ ($f(x)$, $t(x)$) ограничена снизу нулем. Следовательно,

$$\exists n, \forall i \geq n : d(x_i) = d(x_{i+1}) \left(f(x_i) = f(x_{i+1}), t(x_i) = t(x_{i+1})\right),$$

т.е. $A(x_n) = x_n$ и x_n – неподвижная точка.

Реальный объем программы *V* – количество информации (бит), необходимое для записи программы на выбранном языке программирования. Минимальный объем программы V_i – минимальное количество информации (бит), необходимое для записи программы без учета конкретного языка программирования. Очевидно, $V_l = \min V$, где $\{L\}$ – множество языков программирования. Если реальный объем равен минимальному объему (количество операндов равно количеству входных-выходных переменных и количество типов операторов равно двум – существующая, реализующая цель программы и присваивания результатов функции выходным переменным), то стационарная точка существует (данная программа минимальна по *f* и по *d*).

Рассмотрим случай, когда реальный объем больше минимального. Выберем множество P_m из *P* такое, что для любой программы из P_m при любом уменьшении количества операндов увеличивается количество типов операторов и, наоборот, при уменьшении количества типов операторов увеличивается количество операндов. Докажем, что такое непустое P_m существует.

Пусть A^1 – уменьшение операндов программы на единицу, а A^2 – уменьшение типов операторов программы на единицу. Процедура действия оператора A^1: выбираем переменную, которая не является входной-выходной, находим ее значение (текущее значение переменной – «формулу») и заменяем все вхождения переменной-операнда на ее выражение. Процедура действия A^2: если возможно, выражаем выбранный оператор через остальные используемые операторы и переменные, затем подставляем найденное выражение вместо выбранного оператора, если это невозможно – конец.

Для программы $x \in P$ составим последовательность $\{x_i\}$: $x_{i+1} = A(x_i)$, $x_0 = x$. Так как $d(x_i)$ убывает и $d(x_i)$ ограничено снизу, то $\{x_i\}$ сходится к некоторой программе *X*. Объединим такие программы *X* во множество P^* (оно не пустое, оператор A^1 существует для любого x_i). Для программы $y \in P^*$ составим последовательность $\{y_i\}$: $y_{i+1} = A^2(y_i)$, $y_0 = y$. Так как $f(y_i)$ убывает и $f(y_i)$ ограничено снизу, то $\{y_i\}$ сходится к некоторой программе *Y*. Объединение таких программ во множество и есть множество P_m. Конструктивное построение P_m и A^2 для любой программы y_i дает, что множества не пусты. Теперь P_m принадлежит *P* и для любого *x* и *P*: $A(x) = y$, где $y \in P_m$, т.е. $P_m = inf P_m$ из определений A^1, A^2, P_m.

Рассмотрим уравнение $A(x) = y$ и функцию $V(x)$ получения какой-либо характеристики программы x. Следовательно, $V(y)$ ограничено (снизу – нулем, сверху – $V(x)$), как и оператор *A*. Если вход-выход постоянен (для любых программ *x*, *y* из любого *D* входного множества $x(D) = y(D)$), то оператор А монотонен, так как в этом случае $A(x) = A(y)$. Иначе, например, если используются различные алгоритмы, то оператор *A* – не монотонен.

Оператор линейный по функции (цели), нелинейный по структуре: $A(x + y) = A(x) + A(y)$, $A(B(x)) = B(A(x))$. Если фрагмент N_i принадлежит и *x*, и *y*, то в программе $A(x + y)$ он присутствует только один раз, а в программе $A(x) + A(y)$ два раза (в $A(x)$ и в $A(y)$).

Обратный оператор не существует. Корректность, надежность, структурированность, тестируемость и другие параметры программы зависят от сложности программы. Она определяется различными способами. Если Ac_j – оператор минимизации сложности программы, то $Ac_j(z) = Bc_j(z)$ означает, что по критерию C_j результаты действия операторов Ac_j и Bc_j на программу *z* одинаковы, т.е. $C_j(Ac_j(z)) = C_j(Bc_j(z))$.

Если оператор А – монотонный, отображает полную структуру программ *P* в себя, то существует его неподвижная точка *a:* $A(a) = a$.

Действительно, $B = \{b : b \in P, b \leq A(b)\} \neq \emptyset$ в силу полноты (существования нуля из *B*). Следовательно, $\exists a = \sup B$, $\forall b \in P$: $A(a) \geq A(b) \geq b$, поэтому $A(A(a)) \geq A(a)$. Отсюда, $A(a) \in P$, $a \geq A(a)$. Из последних неравенств: $A(a) = a$.

Сложность программ (программных структур) и ее оценка

Сложность системы может быть различного типа и формы, но нас в работе интересуют больше формы.

Представление в форме орграфа. Представим логическую схему программы орграфом: вершины – модули, ребра – их связи, $C_1(x)$ – сложность программы, которую примем равной количеству ребер орграфа. Введем оператор Ac_1: $Ac_1(x) = y$, если для любого z из P: $C_1(y) = C_1(z)$.

Любую программу можно структурировать (например, [3, с. 1]), поэтому оператор Ac_1 существует. Он единственен: для C_1 – оптимальной программы x нет модулей с одинаковыми целями, все связи между модулями необходимы и их минимальное количество, т.е. $C_1(x) = \min C_1(y)$, $y \in P$. Поэтому $Ac_1(x) = x$ и неподвижная точка существует.

Сложность структурированной программы меньше сложности неструктурированной, которая не может быть больше максимального количества связей: $C_1 \le \left(N^2 + N\right)/2$, где N – количество модулей в программе. Следовательно, оператор Ac_1 ограничен. Оператор Ac_1 монотонен – для множества программ P результат действия оператора – одна программа. Оператор Ac_1 нелинейный, условие $Ac_1(x + y) = Ac_1 + Ac_1(y)$ не выполняется: если фрагмент N_i принадлежит и программе x и программе y, тогда в программе $Ac_1(x + y)$ он присутствует только один раз, а в программе $Ac_1(x) + Ac_1(y)$ два раза (в $Ac_1(x)$ и в $Ac_1(y)$).

Обратный оператор существует, но не однозначен, достаточно ввести в C_1 – оптимальную программу модуль-«заглушку» и рассмотреть программу z_i, вызывающую новый модуль, тогда $A_i^{-1}\left(z\right) = z_i$, но и $A_i^{-1}\left(z\right) = z$, так как $A_i(z)$ = z и $A_i(z_i)$ = z.

Критерий C_1 и Ac_1 лучше использовать при сложных взаимосвязях модулей, в многомодульных программах.

Представление в форме дерева. Пусть в программе N модулей и S связей, тогда минимальное количество связей модулей равно $N - 1$. Введем сложность $C_2(x)$ как относительную разницу реального и минимального количества связей: $C_2(x) = (S - N + 1)/(N - 1)$ и оператор минимизации сложности Ac_2: $Ac_2(x) = y$, $\forall z \in P$: $C_2(y) \le C_2(z)$. Оператор Ac_2 приводит структуру программы к древовидному виду. Для согласования связей возможно введение новых модулей.

Возьмем произвольную программу x, подействуем Ac_2, получим $y = Ac_2(x)$ – программу с древовидной структурой. Тогда $y = Ac_2(y)$ и программа y – стационарная точка оператора Ac_2. Для любой программы x $C_2(Ac_2(x)) = 0$, поэтому оператор ограничен снизу нулем; сверху ограничен значением $C_2(x)$ и, следовательно, оператор Ac_2 ограничен. Для любой программы x из множества программ P, достигающих одну цель, результат действия оператора Ac_2 – одна программа, и если $z \in P$, $y = Ac_2(z)$, то тогда $\forall x \in P$: $y = Ac_2(x)$.

Оператор Ac_2 линеен, имеет обратный оператор, но он не однозначный. Критерий C_2, оператор Ac_2 эффективны, когда программа небольшая, необходимо вводить новые модули, она подвержена частым модификациям, содержит сложную обработку данных при сравнительно простых логических связях модулей.

Представление вектор-функцией. Положим $B_{ik} = 1$, если в i-м модуле определяется (используется) k-я глобальная переменная; иначе – $B_{ik} = 0$. Обозначим $F(x)$ – вектор сложности программы, вектор-функция, длина которой равна количеству глобальных переменных. Пусть $F(x)_k$ равно сумме B_{ik} по всем i. Сложность программы $C_3(x)$ оценим суммой $F(x)_k$ по всем k. Введем такой оператор Ac_3, что $Ac_3(x) = y$ и $\forall z \in P$: $C_3(y) \le C_3(z)$.

Существование, ограниченность оператора Ac_3 следует из принципов ООП, причем $\forall x, y \in P$: $C_3(x) \le C_3(y)$ получим: $C_3(Ac_3(x)) = 0$, $C_3(Ac_3(y)) = 0$, $C_3(Ac_3(x)) \le C_3(Ac_3(y))$. Оператор – линейный, обратимый (не однозначно). Критерий C_3 лучше использовать для написания программ на процедурных языках, он показывает, как хорошо задействованы языковые возможности структурирования программ. У хороших программ $C_3(x)$ равно примерно $3v$, где v – количество глобальных переменных. В объектно-ориентированных языках программирования применение C_3 не имеет смысла: почти все программы по нему оптимальны.

Две программы – эквивалентны (пока без уточнения – функционально или структурно, в общем, – по цели), если у них одинаковые цели, элементы, структура, а также можно конструктивно установить отношение «релевантности» (строго говоря, эквивалентности). Процедура выбора определяет на множестве M бинарное отношение типа «релевантность» (обозначим ≈). Можно распространить локальное отношение «релевантность» на всё многообразие M. Оно удовлетворяет свойствам (аксиомам) рефлексивности, симметричности, транзитивности, однозначности (две программы одинаковые по цели, результату – релевантны), непрерывности (мало отличающиеся по цели, структуре программы, незначительно отличаются по релевантности).

Для построения «дерева релевантности» можно применить следующую процедуру. Входные данные: непустые множества – программ P, буфер B, пустой изначально путь в дереве D (список). Процедура:

repeat
удаление из P программ, не отвечающих критериям релевантности D, помещая их в B;
добавление в P и удаление из В программ, удовлетворяющих D;
if($D \neq \emptyset$) *then if* (значения последнего атрибута в D обработаны)
then изменить в D значение последнего параметра следующим
else удалить последний атрибут из D;
if (все программы из одного класса)
then добавить к D лист, содержащий идентификатор класса;
else определяется атрибут с наименьшей мерой релевантности;
добавить в D значение, наилучшее по релевантности;
until ($D = \emptyset$).

Метрик программ – много [4, с. 53]. Их можно разбить на классы:

- по сложности (например, информационной, структурной);
- надежности (устойчивые, самовосстанавливающиеся и др.);
- производительности программы;
- производительности программирования [5, с. 162];
- уровню языка (микрооперации, высокого уровня, запросов и др.);
- модифицируемости (кросс-платформенности, модульности) и др.

Часто рассматриваются меры, базирующиеся на понятии цели, например мера А. Харкевича (причинно-следственной обусловленности): если p_o, p_1 – вероятности достижения критерия до и после получения информации о состоянии системы, то $E = k\left(log_2 p_1 - log_2 p_0\right)$. Применимы также общесистемные меры Н. Моисеева (мера релевантного управления), меры, учитывающие привлекаемые ресурсы типа Шеннона – Уивера, Маргалефа, Симпсона, Макинтоша и др. Широко применимы и проблемно-ориентированные меры – метрики Холстеда, Джилба, Майерса и др.

Метрики «опасны», как любая статистика по данным, распределение которых априори неизвестно полностью. Сотни метрик – разнородные, оценивают ПО, условия разработки, отклонения от норм, спецификаций и др. Наиболее важными являются метрики сложности (информационной, структурной, верификационной, языковой, прогнозной).

Аксиоматизация меры m сложности модульных структур $M = \{M_i : i = 1,2,\ldots,n\}$ может быть проведена на основе аксиом, аналогичных общесистемным, общефункциональным [6, с. 132]:

1) неотрицательность вводимой меры –

$$\forall i = 1,2,\ldots,n : m\left(M_i\right) \geq 0;$$

2) существование программы с нулевой мерой –

$$\exists M_0 \subset M, m\left(M_0\right) = 0;$$

3) мера программы меньше меры всего комплекса –

$$\forall i = 1,2,\ldots,n : m\left(M_i\right) \leq m\left(M\right);$$

4) мера параллельно выполняемого комплекса –

$$m\left(\oplus_{i=1}^{n} M_i\right) = \max m\left(M_i\right);$$

5) мера последовательно выполняемого комплекса –

$$m\left(\otimes_{i=1}^{n} M_i\right) \leq \sum_{i=1}^{n} m\left(M_i\right);$$

6) мера сложности вызова модуля –

$$m\left(M_1 \to M_2\right) \geq m\left(M_1 \oplus M_2\right) - m\left(M_1\right) - m\left(M_2\right),$$

где $M_1 \to M_2$ – вызов модулем M_1 модуля M_2.

Выводы

В работе получены математические результаты (существование, единственность, релевантность, сложность), которые помогут формализовать и структурировать оценочную систему в программировании, провести верификацию, повысить релевантность и снизить алгоритмическую сложность программ. Метрическая теория программ – инструментарий практического исследования метрических качеств программ, верификации программ. Проведена аксиоматизация формальной сложности

программ. Она может быть использована при построении формальной теории оценивания сложности программ в соответствующих пространствах программ.

Список литературы

1. Оптимизация программ и «отладочные» уравнения в метрических пространствах / З.О. Бесланеев [и др.] // Современные наукоемкие технологии. – 2017. – № 5. – С. 13–17.

2. Вареница В.В. Проблема вычисления метрик сложности программного обеспечения при проведении аудита безопасности кода методом ручного рецензирования / В.В. Вареница // Вестник МГТУ им. Н.Э. Баумана, сер. «Приборостроение». – 2011. – С. 79–84.

3. Азаров А.В., Рыбанов А.А. Автоматизированная система расчета метрических характеристик физической схемы базы данных с целью оценки трудоемкости процесса проектирования // Современная техника и технологии. – 2014. – № 5. URL: http://technology.snauka.ru/2014/05/3812 (дата обращения: 05.05.2018).

4. Кайгородцев Г.И. Введение в курс метрической теории и метрологии программ. – Новосибирск: Изд-во НГУ, 2011. – 192 с.

5. Подловченко Р.И. Конечные автоматы в теории алгебраических схем программ / Р.И. Подловченко // Труды Института системного программирования РАН. – 2015. – Т. 27, № 2. – С. 161–172.

6. Треногин В.А. Функциональный анализ: в 2 т. (учебное пособие для вузов) / В.А. Треногин, Б.М. Писаревский, Т.С. Соболева. – М.: Академия, 2013. – Т. 2. – 231 с.