УДК 004.415.532.3

Формальная верификация циклических программ

О.И.Федяев
Донецкий национальный технический университет
fedyaev@donntu.org

***Федяев О.И. Формальная верификация циклических программ. В** статье рассматривается аналитический подход к оценке корректности сложных программ. На примере доказательства правильности программы сортировки проведена апробация теоретико-функционального метода верификации. Формализм программных функций позволил математически строго преодолеть сложности доказательства правильности обработки элементов массива во вложенных циклах. Результаты статьи подтверждают возможность применения данного метода на практике в программной инженерии.*

***Ключевые слова:** верификация программы, корректность программы, программная функция, доказательство правильности программы, теоретико-функциональный подход.*

Введение

Важную роль в обеспечении качества создаваемого программного продукта играет формальная верификация программ [1,2]. Благодаря математическому доказательству корректности программного кода, т.е. его соответствия спецификации решаемой задачи, верификация повышает уровень логической строгости инженерии программного обеспечения [3,4,5].

Среди существующих подходов к формальной верификации выделяется своей оригинальностью и чёткой логикой теоретико-функциональный метод доказательства правильности структурированных программ, который может успешно применяться для проверки корректности циклических программ, использующих массивы [6].

В этом методе правильность программы определяется как соответствие между программой P и её заданной функцией f. Алгебраическая структура исходной программы P характеризует её как составную, что даёт возможность декомпозировать P на элементарные составляющие. Это позволяет задачу верификации программы P свести на основании аксиомы замещения к проверке правильности элементарных подпрограмм P_i, из которых она состоит. Верификация правильности элементарных программ без циклов осуществляется с помощью анализа соответствующих E - схем. Лемма о переходе от итеративных программ к рекурсивным позволяет свести задачу верификации циклических базовых программ P_i (типа while, until, for) к задаче *верификации функционально эквивалентных программ* без циклов. Для вывода программных функций $[P_i]$ используются трассировочные таблицы и разделяющиеся условные правила. В данном методе проблема перестановки элементов решается детальным развёртыванием и свёртыванием изменяющихся частей массива. Формально полная правильность программы P состоит в нахождении программной функции [P] и сравнении её с заданной функцией f , т.е. в проверке равенства $f = [P]$.

В данной работе ставится задача оценить эффективность данного метода на примере формальной верификации программы сортировки по убыванию элементов массива методом выбора [7]. Алгоритм сортировки на рис.1 составлен специально так, чтобы в цикле присутствовали перестановки элементов массива, что делает задачу верификации не тривиальной. Процесс верификации программы P включает последовательность шагов доказательства правильности элементарных программ, из которых состоит P.

Шаг 1. Верификация подпрограммы типа «if-then-else»

В блок-схеме алгоритма сортировки на рис.1 первой элементарной программой (обозначим её P_1), с которой надо начать проверку правильности, является «внутренний» условный оператор (рис.2).

Поскольку функция f_1 для программы P_1 не задана, то выдвинем гипотезу о ней и запишем её в виде следующего предложения одновременного присваивания [6]

$$f_1 = \left(i, a_j, a_i := i + 1, \max(a_j, a_i), \min(a_j, a_i)\right) \quad (1)$$

Эта запись означает, что в начале вычисляются значения всех выражений, стоящих справа от символа «:=», а затем полученные значения присваиваются соответствующим именам данных, стоящим слева. Правильность P_1 следует из доказательства равенства $f_1 = [P_1]$. Поэтому найдём программную функцию $[P_1]$ с помощью трассировочной таблицы.

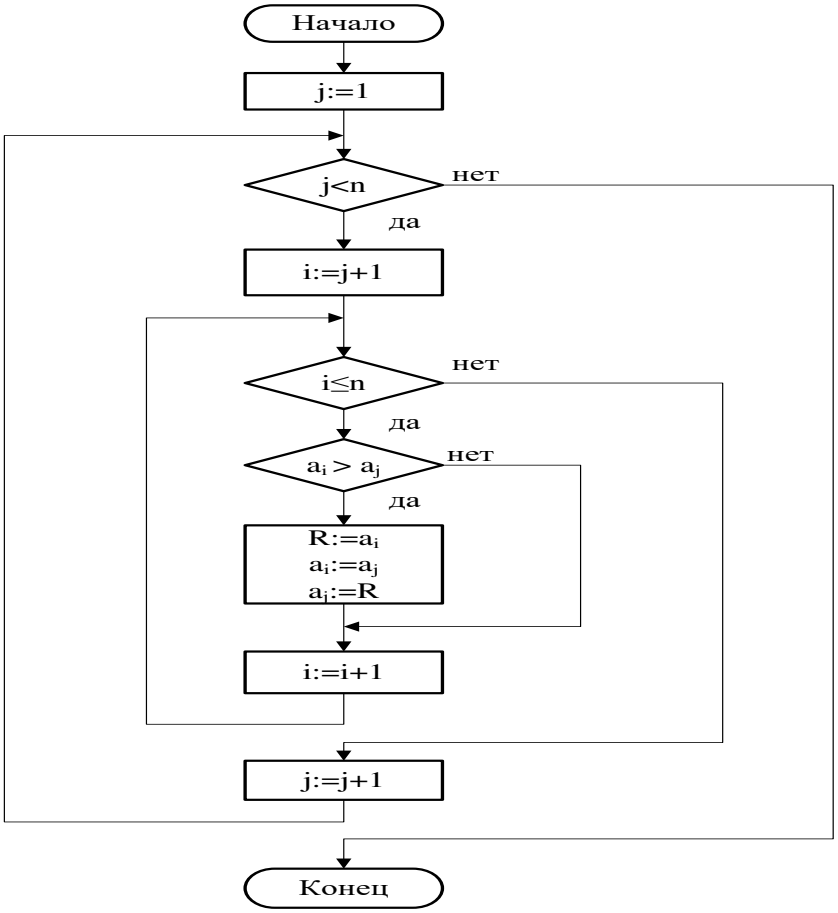


Рисунок 1 - Блок-схема программы P сортировки массива

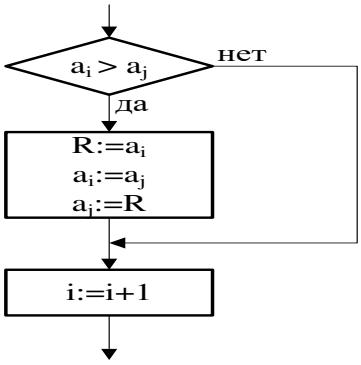


Рисунок 2 - Элементарная программа P_1 типа «if-then-else»

Трассировочная таблица позволяет формально записать вывод функции условно последовательной программы P_1 . Трассировка первого пути выполнения программы P_1 показана в табл. 1 .

Таблица 1. Трассировка пути выполнения P_1 , когда $(a_i > a_j) = \mathbf{true}$

N п/п	Фрагмент	a_j	a_i	R	i
1	$a_i > a_j$	$a_j^1 = a_j^0$	$a_i^1 = a_i^0$	$R^1 = R^0$	$i^1 = i^0$
2	$R := a_i$	$a_j^2 = a_j^1$	$a_i^2 = a_i^1$	$R^2 = a_i^1$	$i^2 = i^1$
3	$a_i := a_j$	$a_j^3 = a_j^2$	$a_i^3 = a_j^2$	$R^3 = R^2$	$i^3 = i^2$
4	$a_j := R$	$a_j^4 = R^3$	$a_i^4 = a_i^3$	$R^4 = R^3$	$i^4 = i^3$
5	$i := i + 1$	$a_j^5 = a_j^4$	$a_i^5 = a_i^4$	$R^5 = R^4$	$i^5 = i^4$+1

Если просматривать снизу вверх соответствующие колонки таблицы и при этом систематически исключать все промежуточные значения верхних индексов, начиная с конечных значений, то получим программную функцию для первого пути, исключив побочное данное R:

– изменение данных

$a_j^5 = a_j^4 = R^3 = R^2 = a_i^1 = a_i^0$;

$a_i^5 = a_i^4 = a_i^3 = a_j^2 = a_j^1 = a_j^0$; $i^5 = i^4 + 1 = i^3 + 1 = i^0 + 1$;

– программная функция пути

$(a_i > a_j) \to i, a_j, a_i := i + 1, a_i, a_j$. (2)

Выполнив аналогично трассировку второго пути выполнения программы P_1 , когда $(a_i > a_j =$ **false**$)$, получим другую часть программной функции, соответствующую второму пути:

$(a_i \le a_j) \to i, a_j, a_i := i + 1, a_j, a_i$. (3)

Объединение (2) и (3) даёт общую программную функцию $[P_1]$ в виде условного правила

$[P_1] = (a_i > a_j) \to i, a_j, a_i := i + 1, a_i, a_j \mid$

$(a_i \le a_j) \to i, a_j, a_i := i + 1, a_j, a_i$,

которое можно записать в более компактной форме

$[P_1] = \left(i, a_j, a_i := i + 1, \max(a_j, a_i), \min(a_j, a_i)\right)$. (4)

Сравнивая (1) и (4) видно, что $f_1 = [P_1]$ и, следовательно, высказанная гипотеза о программной функции верна.

Шаг 2. Верификация подпрограммы типа «while»

На основании аксиомы замещения [6] можно выполнить замену подпрограммы P_1, правильность которой доказана на шаге 1, одним оператором с той же программной функцией f_1. При этом программная функция исходной программы P не изменится. После такой подстановки выделяется следующий элементарный структурный элемент для верификации, который является циклом типа «while» (рис. 3).

Как видно из рисунка, тело цикла представлено одним оператором, который реализует функцию f_1. Исходная функция, которую должна реализовать верифицируемая программа P_1 , не задана. Поэтому сначала сформулируем гипотезу о функции для элементарной циклической программы типа **while** (рис. 3). Для этого достаточно рассмотреть три шага работы оператора цикла, начиная с номера *i=j+1*.

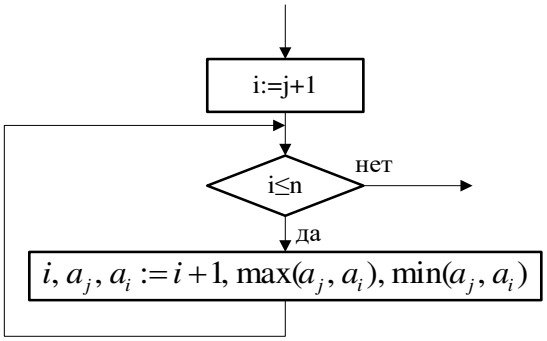
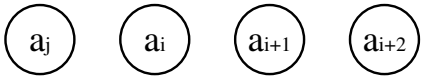


Рисунок 3 - Элементарная программа P_2 типа «while»

Перестановка элементов в группе из четырёх последовательных элементов массива



будет определяться следующими операторами одновременного присваивания, которые показывают, как изменяются значения элементов в этих позициях на каждом цикле в зависимости от исходных значений элементов перед началом циклического процесса:

1 шаг: $a_j^1, a_i^1, a_{i+1}^1 a_{i+2}^1 := \max(a_j, a_i), \min(a_j, a_i)$, a_{i+1}, a_{i+2} ;

2 шаг: $a_j^2, a_i^2, a_{i+1}^2 a_{i+2}^2 := \max(\max(a_j, a_i), a_{i+1}), \min(a_j, a_i)$, $\min(\max(a_j, a_i), a_{i+1}), a_{i+2}$;

3 шаг: $a_j^3, a_i^3, a_{i+1}^3 a_{i+2}^3 := \max(\max(\max(a_j, a_i), a_{i+1}), a_{i+2}),\ \min(a_j, a_i)$, $\min(\max(a_j, a_i), a_{i+1}), \min(\max(\max(a_j, a_i), a_{i+1}), a_{i+2})$.

Из этих шагов видна закономерность, которую можно записать в виде следующей функции:

$$f_2 = (j < i < n) \to i, a_j, a(i{:}n) := n+1, \max\Big[a_j, \max[a(i{:}n)]\Big],$$
$$\boxed{i} \quad \min[a_i, a_j],$$
$$\boxed{i+1} \quad \min\Big[a_{i+1}, \max[a_j, a(i{:}i)]\Big],$$
$$\boxed{i+2} \quad \min\Big[a_{i+2}, \max[a_j, a(i{:}i+1)]\Big],$$
$$\ldots$$
$$\boxed{n} \quad \min\Big[a_n, \max[a_j, a(i{:}n-1)]\Big].$$

В прямоугольных рамках указаны номера элементов массива. Теперь докажем правильность следующей циклической программы

$P_2 =$ **while** $i \le n$ **do** $i, a_j, a_i :=$

$i+1, \max(a\_j, a\_i), \min(a\_j, a\_i)$ **od** ,

где ключевые слова **do** и **od** являются обычными ограничителями фрагментов программы.

Используя лемму о рекурсивном представлении [6], заменим программу P_2 на рекурсивную нециклическую программу P_2' , которая будет иметь следующий вид:

$P_2' =$ **if** $i < n$

then $i, a_j, a_i := i+1, \max(a_j, a_i), \min(a_j, a_i)$;

$(j < i < n) \to i, a_j, a(i{:}n) := n+1, \max\Big[a_j, \max[a(i{:}n)]\Big],$
$\min[a_i, a_j],$
$\min\Big[a_{i+1}, \max[a_j, a(i{:}i)]\Big],$
$\min\Big[a_{i+2}, \max[a_j, a(i{:}i+1)]\Big],$
$\ldots$
$\min\Big[a_n, \max[a_j, a(i{:}n-1)]\Big];$

fi .

Докажем справедливость равенства $f_2 = [P_2']$, что эквивалентно доказательству $f_2 = [P_2]$. Найдём с помощью трассировочной таблицы программную функцию $[P_2']$.

Таблица 2. Трассировочная таблица для программы P_2'

Фрагмент программы	Условие	Массив a	i
1	$i_0 \le n$	$a_1(1{:}n) = a_0(1{:}n)$	$i_1 = i_0$
2	$j < i_1 < n$	(2.1) $a_2(1{:}j-1) = a_1(1{:}j-1)$ (2.2) $a_2(j) = \max[a_1(j), a_1(i_1)]$ (2.3) $a_2(j+1{:}i_1-1) = a_1(j+1{:}i_1-1)$ (2.4) $a_2(i_1) = \min[a_1(j), a_1(i_1)]$ (2.5) $a_2(i_1+1{:}n) = a_1(i_1+1{:}n)$	$i_2 = i_1 + 1$
3	$j < i_2 < n$	(3.1) $a_3(1{:}j-1) = a_2(1{:}j-1)$ (3.2) $a_3(j) = \max\big[a_2(j), \max[a_2(i_2{:}n)]\big]$ (3.3) $a_3(j+1{:}i_2-1) = a_2(j+1{:}i_2-1)$ (3.4) $a_3(i_2) = \min[a_2(i_2), a_2(j)]$ (3.5) $a_3(i_2+1) =$ $\min\big[a_2(i_2+1), \max[a_2(j), a_2(i_2{:}i_2)]\big]$ $\ldots$ $a_3(n) = \min\big[a_2(n), \max[a_2(j), a_2(i_2{:}n-1)]\big]$	$i_3 = n+1$

Условие:

$$(i_0 \le n) \wedge (j < i_1) \wedge (i_1 < n) \wedge (j < i_2) \wedge (i_2 < n) =$$
$$(i_0 \le n) \wedge (j < i_0) \wedge (i_0 < n) \wedge (j < i_0 + 1) \wedge (i_0 + 1 < n) =$$
$$(j < i_0 \le n) \wedge (j < i_0 + 1 < n) = (j < i_0 < n)$$

Если проследить за изменением данных по трассировочной таблице путём систематического исключения всех промежуточных значений $a_3(1{:}\,j-1) = a_2(1{:}\,j-1) = a_1(1{:}\,j-1) = a_0(1{:}\,j-1)$ индексов, то получим функцию рекурсивной программы. Из формулы (3.1) в трассировочной таблице (см. табл. 2) выводим:

$$a_3(1{:}\,j-1) = a_2(1{:}\,j-1) = a_1(1{:}\,j-1) = a_0(1{:}\,j-1)$$

Таким же способом из формулы (3.2) получаем:

$$a_3(j) = \max\big[\max[a_1(j), a_1(i_1)], \max[a_2(i_1+1{:}\,n)]\big]$$
$$= \max\big[\max[a_0(j), a_0(i_0)], \max[a_1(i_0+1{:}\,n)]\big]$$
$$= \max\big[\max[a_0(j), a_0(i_0)], \max[a_0(i_0+1{:}\,n)]\big]$$
$$= \max[a_0(j), \max[a_0(i_0{:}\,n)]]$$

Из формулы (3.3) получаем:

$$a_3(j+1{:}\,i_2-1) = a_2(j+1{:}\,i_2-1)$$
$$a_3(j+1{:}\,i_1+1-1) = a_2(j+1{:}\,i_1+1-1)$$
$$a_3(j+1{:}\,i_1) = a_2(j+1{:}\,i_1) = a_1(j+1{:}\,i_1), \min[a_1(j), a_1(i_1)]$$
$$a_3(j+1{:}\,i_0) = a_2(j+1{:}\,i_0) = a_0(j+1{:}\,i_0-1), \min[a_0(j), a_0(i_0)]$$
$$a_3(j+1{:}\,i_0-1) = a_0(j+1{:}\,i_0-1)$$
$$a_3(i_0) = \min[a_0(j), a_0(i_0)]$$

Из формулы (3.4) получаем:

$$a_3(i_2) = \min[a_2(i_2), a_2(j)]$$
$$a_3(i_1+1) = \min[a_2(i_1+1), a_2(j)]$$
$$a_3(i_0+1) = \min\big[a_1(i_1+1), \max[a_1(j), a_1(i_1)]\big]$$
$$= \min\big[a_0(i_0+1), \max[a_0(j), a_0(i_0)]\big]$$
$$= \min\big[a_0(i_0+1), \max[a_0(j), a_0(i_0{:}\,i_0)]\big]$$

Из формулы (3.5) получаем:

$$a_3(i_2+1) = \min\big[a_2(i_2+1), \max[a_2(j), a_2(i_2{:}\,i_2)]\big]$$
$$a_3(i_0+2) = \min\big[a_2(i_1+2), \max[a_2(j), a_2(i_1+1{:}\,i_1+1)]\big]$$
$$= \min\big[a_1(i_1+2), \max[\max[a_1(j), a_1(i_1)], a_2(i_1+1{:}\,i_1+1)]\big]$$
$$= \min\big[a_0(i_0+2), \max[a_0(j), a_0(i_0{:}\,i_0+1)]\big] \quad \text{и т. д.}$$

Если объединить результаты выводов по формулам (3.1) – (3.5) из табл. 2 и отбросить нулевые индексы, то получим программную функцию рекурсивной программы P_2'. Она полностью совпадает с выдвинутой гипотезой f_2, т.е. $f_2 = [P_2']$. Принимая во внимание лемму о рекурсивном представлении программы и теорему правильности [3], можно утверждать, что циклическая программа P_2 также правильна.

Шаг 3. Верификация программы типа «последовательность»

Далее переходим к доказательству правильности следующей элементарной программы P_3 (последовательность), которая получается после применения аксиомы замещения правильного цикла **while** его функцией (рис.4). Выдвинем следующую гипотезу о функции программы P_3 :

$$f_3 = (j < n) \to i, a(j), a(j+1{:}n) := n+1,$$
$$\max[a(\mathrm{j{:}\,n})],$$
$$\min\big[a(j+1), \max[a(j{:}j)]\big],$$
$$\min\big[a(j+2), \max[a(j{:}j+1)]\big],$$
$$\min\big[a(j+3), \max[a(j{:}j+2)]\big],$$
$$\cdots$$
$$\min\big[\ \mathrm{a(n)}\ , \max[\mathrm{a(j{:}\,n-1)}]\big]. \qquad (5)$$

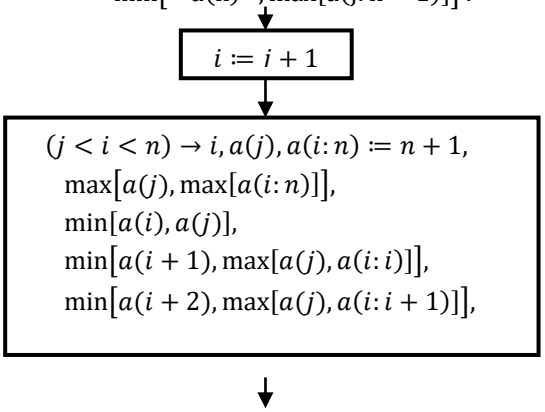


Рис. 4. Элементарная программа P_3 типа «последовательность»

Упростим запись функции f_3 , представив её в виде

$$f_3 = (j < n) \to i, a(j), a(j+1{:}n) := n+1, \max[a(j{:}n)], a(j+1{:}n)\backslash a(j).$$

Символ «\» обозначает операцию исключения элементов. Поэтому запись $a(k{:}n)\backslash a(j{:}k)$ означает, что в массиве $a(j{:}n)$ среди элементов $a(k{:}n)$ нет элементов этого же массива, стоящих на позициях от j до k ($j<k<n$). Такое расположение элементов массива возникает после перестановки максимальных элементов, т.е. максимальные элементы «всплыли» в левую часть массива $a(j{:}k)$, а меньшие элементы переместились в позиции $a(k{:}n)$.

Представленную на рис. 4 последовательность из двух операторов можно рассматривать как трассировочную таблицу. В этом случае, если во втором операторе блок-схемы (т.е. в функции f_2) заменить i на j+1, то получится программная функция [P_3] , которая будет совпадать с f_3. Таким образом, из равенства $f_3 = [P_3]$ следует правильность программы P_3.

Шаг 4. Верификация подпрограммы типа «while»

Последующая декомпозиция структуры исходной программы P даёт следующий простой фрагмент для верификации – цикл типа «while», представленный на рис. 5. Тело этого цикла получено на основании аксиомы замещения, путём замены правильной последовательности P_3 на функцию f_3.

Сформулируем гипотезу для программы P_4 в виде функции f_4 :

$$(j < n) \to j, i, a(j{:}n) := n, n+1, \max[a(j{:}n)],$$
$$\max[a(j+1{:}n)\backslash a(j)],$$
$$\max[a(j+2{:}n)\backslash a(j{:}j+1)],$$
$$\max[a(j+3{:}n)\backslash a(j{:}j+2)],$$
$$\cdots$$
$$\max[\ a(n{:}n)\ \backslash\ a(j{:}n-1)]. \qquad (6)$$

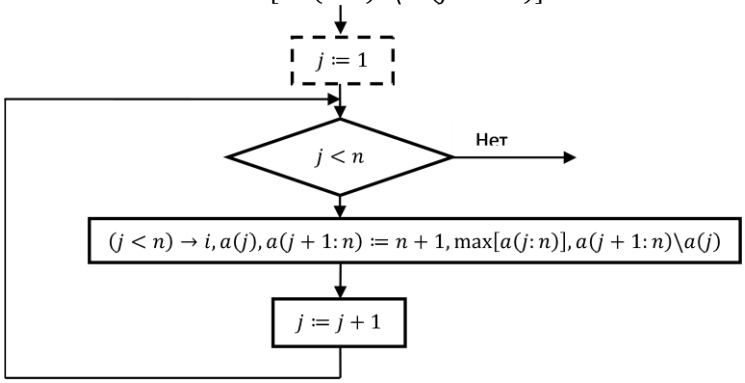


Рисунок 5 - Элементарная программа P_4 типа «while»

$$P_4 = \textbf{while } j < n \textbf{ do } (j < n) \to i, a(j), a(j+1:n) := n+1,$$
$$\max[a(j:n)], a(j+1:n) \backslash a(j);$$
$$j := j+1;$$
$$\textbf{od};$$

Далее, используя лемму о рекурсивном представлении [6], представим программу P_4 в рекурсивной ациклической форме P_4', что упростит получение программной функции и саму верификацию:

$$P_4' = \textbf{if } j < n \textbf{ then } (j < n) \to i, a(j), a(j+1:n) := n+1,$$
$$\max[a(j:n)],$$
$$a(j+1:n) \backslash a(j);$$
$$j := j+1;$$
$$(j < n) \to j, i, a(j:n) := n, n+1, \max[a(j:n)],$$
$$\max[a(j+1:n) \backslash a(j:j)],$$
$$\max[a(j+2:n) \backslash a(j:j+1)],$$
$$\ldots$$
$$\max[a(n:n) \backslash a(j:n-1)];$$
$$\textbf{fi}$$

Докажем, что $f_4 = [P_4']$. Для этого выведем из трассировочной табл.3 программную функцию $[P_4']$.

Таблица 3. Трассировка штатного пути выполнения программы P_4'

Фрагмент программы	Условие	Массив $a(1:n)$	j
1	$j_0 < n$	$a_1(1:n) = a_0(1:n)$	$j_1 = j_0$
2	$j_1 < n$	(2.1) $a_2(1:j_1-1) = a_1(1:j_1-1)$ (2.2) $a_2(j_1) = \max[a_1(j_1:n)]$ (2.3) $a_2(j_1+1:n) = a_1(j_1+1:n) \backslash a(j_1)$	$j_2 = j_1$
3		$a_3(1:n) = a_2(1:n)$	$j_3 = j_2 + 1$
4	$j_3 < n$	(4.1) $a_4(1:j_3-1) = a_3(1:j_3-1)$ (4.2) $a_4(j_3) = \max[a_3(j_3:n)]$ (4.3) $a_4(j_3+1) = \max[a_3(j_3+1:n) \backslash a(j_3:j_3)]$ (4.4) $a_4(j_3+2) = \max[a_3(j_3+2:n) \backslash a(j_3:j_3+1)]$ . . . $a_4(n) = \max[a_3(n:n) \backslash a(j_3:n-1)]$	$j_4 = n$

Проследим только за изменением элементов массива a и параметра j :

$$a_4(1:j_3-1) = a_3(1:j_3-1)$$
$$a_4(1:j_2) = a_3(1:j_2)$$
$$a_4(1:j_0) = a_2(1:j_2) = a_2(1:j_1) = a_1(1:j_1-1), \max[a_1(j_1:n)]$$
$$= a_0(1:j_0-1), \max[a_0(j_0:n)]$$

$$a_4(j_3) = \max[a_3(j_3:n)]$$
$$a_4(j_2+1) = \max[a_2(j_2+1:n)] = \max[a_2(j_1+1:n)]$$
$$a_4(j_1+1) = \max[a_1(j_1+1:n) \backslash a(j_1)]$$
$$a_4(j_0+1) = \max[a_0(j_0+1:n) \backslash a(j_0)]$$

$$a_4(j_3+1) = \max[a_3(j_3+1:n) \backslash a(j_3:j_3)]$$
$$a_4(j_2+2) = \max[a_2(j_2+2:n) \backslash a(j_2+1:j_2+1)]$$
$$a_4(j_1+2) = \max[a_2(j_1+2:n) \backslash a(j_1+1:j_1+1)]$$
$$= \max[a_1(j_1+2:n) \backslash a(j_1) \backslash a(j_1+1:j_1+1)]$$
$$a_4(j_0+2) = \max[a_0(j_0+2:n) \backslash a(j_1:j_1+1)]$$
$$= \max[a_0(j_0+2:n) \backslash a(j_0:j_0+1)]$$
$$\ldots$$

$$\begin{aligned} a_4(n) &= \max[a_3(n{:}n)\backslash a(j_3{:}n-1)] \\ &= \max[a_2(n{:}n)\backslash a(j_2+1{:}n-1)] \\ &= \max[a_1(n{:}n)\backslash a(j_1)\backslash a(j_1+1{:}n-1)] \\ &= \max[a_0(n{:}n)\backslash a(j_1{:}n-1)] \\ &= \max[a_0(n{:}n)\backslash a(j_0{:}n-1)] \end{aligned}$$

$j_4 = n.$

Из анализа изменения элементов массива видно, что перемещались только те элементы, которые расположены после j-го номера. Причём, функционально изменение полностью совпадают с выдвинутой гипотезой. Если рассмотреть также и другой путь выполнения программы P_4, связанный с ложностью выполнения условия $(j_3 < n) = \text{false}$, то совпадать будут не только данные, но и предикат программной функции. Отсюда следует выполнение равенства $f_4 = [P_4']$ и правильность цикла P_4.

Шаг 5. Верификация программы типа «последовательность»

На последнем этапе процесс верификации сводится к доказательству правильности элементарной программы (рис.6), полученной путём свёртки по аксиоме замещения последовательного ряда доказанных ранее правильных подпрограмм. Гипотезу о программной функции f_5, реализуемую программой P_5, представим в следующем виде:

$$\begin{aligned} f_5 = (1 < n) \to j, i, a(1{:}n) \coloneqq n, n+1, &\max[a(1{:}n)], \\ &\max[a(2{:}n)\backslash a(1)], \\ &\max[a(3{:}n)\backslash a(1{:}2)], \\ &\quad \cdots \\ &\max[a(n{:}n)\backslash a(1{:}n-1)]. \end{aligned}$$

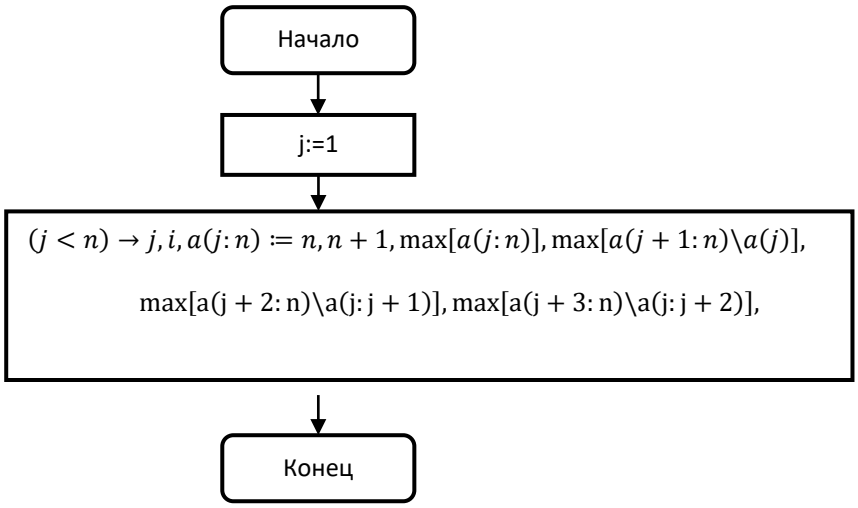


Рисунок 6 - Элементарная программа P_5 типа «последовательность»

Последовательность операторов программы, приведенная на рис.6, позволяет легко получить программную функцию $[P_5]$ без трассировочной таблицы. Для этого достаточно во втором операторе (функции f_4) заменить переменную j на константу 1. Полученная таким образом программная функция полностью совпадает с гипотезой, т.е. $f_5 = [P_5]$, что подтверждает правильность P_5 и, следовательно, правильность исходной программы P.

Поскольку значения функции max удовлетворяют следующему неравенству

$$\max[a(1{:}n)] \geq \max[a(2{:}n)\backslash a(1)] \geq \max[a(3{:}n)\backslash a(1{:}2)] \geq \cdots ,$$

то можно утверждать, что элементы массива $a(1{:}n)$ отсортированы по убыванию правильно, т.е. справедливо $a(1{:}n) = \text{SORT}(a(1{:}n))$.

Заключение

Возрастающая сложность и важность

разрабатываемых программ требует серьёзного отношения к обеспечению качества производимого программного продукта.

По этой причине в последние годы сделаны огромные усилия по созданию новых технологий разработки программного обеспечения (ПО), методов и инструментальных средств анализа корректности ПО [8,9]. Традиционное тестирование, опираясь на разнообразные методы и мощные средства автоматизации, однако, не могут исключить применение формальных методов доказательства корректности программ, особенно – критически важных. Наиболее перспективный путь в решении проблемы качества ПО заключается в сочетании методов тестирования и верификации программ в составе специализированных инструментальных CASE-систем [10].

В данной работе выполнена апробация формального метода верификации, основанного на понятии программной функции. На примере верификации достаточно сложной циклической программы обработки массива показана практическая возможность применения данного метода в программной инженерии.

Литература

1. IEEE 1012-2004. Standard for Software Verification and Validation. IEEE, 2005.

2. IEEE 1059-1993. Guide for Software Verification and Validation Plans. New York: IEEE, 1993.

3. Непомнящий В.А., Рякин О.М. Прикладные методы верификации программ / Под ред. А.П. Ершова. – М.: Радио и связь, 1988. – 256 с.

4. Кларк Э.М., Грамберг О., Пелед Д. Верификация моделей программ: Model Checking. Пер. с англ./ Под ред. Р.Смелянского. – М.: МЦНМО, 2002. – 416 с.

5. Вудкок Д. Первые шаги к решению проблемы верификации программ. Открытые системы . – 2006. – № 8. – С. 36-43.

6. Лингер Р. И., Миллс Х., Уитт Б. Теория и практика структурного программирования: Пер. с англ.М.: Мир, 1982. – 406 с.

7. Седжвик Р. Фундаментальные алгоритмы на C++. Анализ, структуры данных, сортировка, поиск: Пер. с англ./ Р. Седжвик. – СПб.: ООО «ДиаСофтЮП», 2002. – 688 с.

8. Beckert B., Hahnle R., Schmitt P.H., eds. Verification of Object-Oriented Software: The KeY Approach. Springer, 2007.

9. Hoare T., Misra J. Verified software: Theories, Tools, Experiments. Vision of Grant Challenge project. Microsoft Research Ltd and the University of Texas at Austin, 2005. – P. 1–43.

10. Васенин В.А.Кривчиков М.А. Языково-ориентированное программирование для формальной верификации программного обеспечения. Материалы четвертой Научно-практической конференции «Актуальные проблемы системной и программной инженерии». Сб. науч. тр. /Национальный исследовательский университет «Высшая школа экономики». – М.: Изд-во НИУ ВШЭ, 2015. -234 с

Федяев О.И. Формальная верификация циклических программ. В с*татье рассматривается аналитический подход к оценке корректности сложных программ. На примере доказательства правильности программы сортировки проведена апробация теоретико-функционального метода верификации. Формализм программных функций позволил математически строго преодолеть сложности доказательства правильности обработки элементов массива во вложенных циклах. Результаты статьи подтверждают возможность применения данного метода на практике в программной инженерии.*

Ключевые слова*: верификация программы, корректность программы, программная функция, доказательство правильности программы, теоретико-функциональный подход.*

Fedyaev O. Formal verification cycle programs. *The article deals with an analytical approach to the assessment of the correctness of complex programs. On the example of the proof of the correctness of the program sorting is carried out check method of the program functions. Formalisms of the program functions allow a mathematically rigorous proof the correctness perform nested loops with arrays. Article results confirm the possibility of using this method in practice in software engineering.*

Keywords: *Verification of the program, program correctness, program function, proof of the correctness of the program, method of the program functions.*

Статья поступила в редакцию 20.11.2016
Рекомендована к публикации д-ром физ.-мат.. наук А.С. Миненко