УДК 51.681.3

А.Н. Максимец

ВЕРИФИКАЦИЯ ПРОГРАММ: СОСТОЯНИЕ, ПРОБЛЕМЫ, ЭКСПЕРИМЕНТАЛЬНЫЕ РЕЗУЛЬТАТЫ. II

Рассматриваются алгоритмы поиска инвариантных соотношений в программах с простыми переменными, которые относятся к методам анализа потоков данных и верификации. Приводится краткий обзор таких методов и примеры для илюстрации работы предлагаемых алгоритмов.

Введение

Данная работа – это продолжение предыдущей (I части) [1].

В первой части работы были представлены краткие сведения о методах верификации и актуальных задач проблемы верификации программ. В ней обозревалась проблематика верификации реактивных и функциональных систем, где указано место предложенного подхода в формальных методах верификации. Приводились основные понятия, используемые в этой области, а также описывались свойства введенных понятий вместе с обоснованием их правильности.

В данной статье более детально рассматриваются методы верификации и их реализация применительно к конкретным программам над конкретными алгебрами данных. Детально рассматривается язык типа равенств для программ над абсолютно свободными алгебрами данных и кольцами полиномов. Приведены результаты экспериментов для программ над этими алгебрами данных.

1. Язык равенств. Основные задачи

Пусть A – $U-Y$-программа с множеством переменных $R=\{r_1,r_2,\ldots,r_m\}$, рассматриваемая над алгеброй данных (D,Ω), $K(\Omega,Eq)$ класс алгебр, содержащий алгебру (D,Ω) и определяемый множеством тождественных соотношений Eq, а $T_D(R)$ – свободная алгебра термов над R из класса $K(\Omega,Eq)$.

Рассмотрим проблему поиска инвариантов для языка L, состоящего из условий типа равенств $g(r)=h(r)$, где $g(r),h(r)\in T_D(R)$, $r=(r_1,\ldots,r_m)$ (т. е. L не учитывает условий из U).

Приведем необходимые определения и некоторые результаты общего характера, следуя работам [2, 4, 5, 6].

Пусть M – некоторое множество равенств. Алгебраическим замыканием множества M относительно Eq называется наименьшее множество $C(M)$, содержащее рефлексивное, симметричное и транзитивное замыкание M, все тождества из Eq и для всякой n-арной операции $\omega\in\Omega$ вместе с парами $(g_1(r),q_1(r)),\ldots,(g_n(r),q_n(r))$ содержит пару $(\omega(g_1(r),\ldots,g_n(r)),\ \omega(q_1(r),\ldots,q_n(r)))$.

Множество M называется алгебраически замкнутым, если $C(M)=M$.

Теорема 1. $C(M)=M\Leftrightarrow M$ – конгруэнция на $T_D(R)$.

Доказательство очевидным образом следует из определений конгруэнции и алгебраического замыкания.

Подмножество P алгебраически замкнутого множества M называется алгебраическим базисом M, если $C(P)=M$.

Пусть M – алгебраически замкнутое множество равенств. Соответствующую фактор-алгебру будем обозначать $T_D(R)/M$, ее элементы $t(modM)$, $t\in T_D(R)$, а равенство термов в виде $t=t'(modM)$. С каждым оператором присваивания $y=(r_1:=t_1(r),\ldots,r_m:=t_m(r))$ $(t_i\in T_D(R))$ и алгебраически замкнутым множеством M свяжем гомоморфизм

$h_y : T_D(R) \to T_D(R)/M$, полагая $h_y(r_i) = t_i(modM)$.

Конгруэнция M на $T_D(R)$ называется нормальной, если она является ядром эндоморфизма алгебры $T_D(R)$, т. е. $T_D(R)/M$ изоморфна подалгебре алгебры $T_D(R)$.

Пусть $ef(M, y)$ (сужение $ef(M, u, y)$) обозначает множество равенств вида $t(r) = t'(r)$ таких, что $t(t_1, \ldots, t_m) = t'(t_1, \ldots, t_m) \in M$.

Очевидно, что $ef(M, y)$ – конгруэнция и $ef(M, y) = ker(h_y)$, где $h_y : T_D(R) \to T_D(R)/M$ – некоторый гомоморфизм.

Лемма 1.

$$ef(ef(M, y), y') = ef(M, yy').$$

Теорема 2. Если M – нормальная конгруэнция, то $ef(M, y)$ тоже нормальная конгруэнция.

Функция ef называется дистрибутивной, если для любых M и M' верно

$$ef(M \bigcap M', y) = ef(M, y) \bigcap ef(M', y).$$

Теорема 3. Если множества M, M' алгебраически замкнуты, то функция ef дистрибутивна.

Заметим, что свойство дистрибутивности функции ef более сильное, чем свойство монотонности, т.к. из дистрибутивности функции ef следует ее монотонность.

Действительно, если $M \subseteq M'$, то в силу дистрибутивности функции ef можно записать

$$ef(M \bigcap M', y) = ef(M', y) = ef(M, y) \bigcap ef(M', y).$$

Откуда следует, что $ef(M, y) \subseteq ef(M', y)$.

Допустим что известно как строить множество $ef(M, y)$ или его алгебраический базис и находить пересечение таких множеств или алгебраический базис этого пересечения. Тогда, используя одну из формул

$$N_a^{(n)} = \bigcap_{(a', u, y, a) \in S} ef(N_{a'}^{(n-1)}, u, y), \quad n > 0, \ a, a' \in A, \tag{1}$$

или

$$N_a^{(n)} = N_a^{(n-1)} \bigcap \Big(\bigcap_{(a', u, y, a) \in S} ef(N_{a'}^{(n-1)}, u, y) \Big), \quad n > 0, \ a, a' \in A, \tag{2}$$

можно организовать процесс поиска инвариантов в состояниях программы, отправляясь от некоторых начальных алгебраически замкнутых множеств, сопоставленных состояниям программы, и повторять его до тех пор, пока множества $ef(M, y)$ в данных состояниях не перестанут изменяться (стабилизируются). Заметим, что если множества $ef(M, y)$ обладают конечными базисами, то из стабилизации множеств $ef(M, y)$ следует стабилизация их базисов и наоборот. Следовательно, проблема построения множеств инвариантов в состояниях U-Y-программы над заданной алгеброй данных $T_D(R)$ сводится к следующим основным задачам.

Задача о соотношениях. Построить по заданному множеству равенств M (или его алгебраическому базису) и оператору $y \in Y$ множество $ef(M, y)$ (или его алгебраический базис).

Задача о пересечении. По заданным множествам $ef(M, y)$ и $ef(M', y)$ построить множество $ef(M, y) \bigcap ef(M', y)$ (или его алгебраический базис).

Задача о стабилизации. Показать, что процесс построения множеств $ef(M, y)$ (или их алгебраических базисов), сопоставленных состояниям программы, стабилизируется.

В определении метода МВА процесс вычисления инвариантов зависит, вообще говоря, от выбора совокупности простых путей, определяющих начальное приближение. Аналогичная ситуация имеет место и при переходе от параллельного

вычисления приближений по формулам (1) и (2), когда на каждом шаге итерации вычисления производятся одновременно для всех состояний, к последовательному, когда в текущий момент для разных состояний рассматриваются, вообще говоря, разные приближения. Последовательное вычисление приближений позволяет существенно уменьшить объем вычислений, так как для части состояний на определенных итерациях вообще не будет происходить изменений. Например, если существует путь из a в a' и не существует пути из a' в a, то множество инвариантов в состоянии a никак не зависит от соответствующего множества для состояния a' и процессы вычисления инвариантов для таких состояний удобно разнести, вычисляя их сначала для a и лишь затем для a'.

Далее приводятся последовательные алгоритмы нижней и верхней аппроксимации.

Относительно исходной U-Y-программы будем предполагать. Во-первых, что все ее состояния, за исключением быть может начального и заключительного, являются ветвлениями или слияниями. Этого можно достичь в результате перемножения операторов присваивания на линейных участках. Во-вторых, что с каждым состоянием a ассоциируется множество соотношений N_a. Пусть S – множество переходов U-Y-программы и

$$Ps(a) = \{a' \in A \mid (a,u,y,a') \in S\}.$$

В описываемых далее алгоритмах N_a, N – переменные типа “множество”, значения которых – множества соотношений в состоянии $a \in A$, N_0 – начальное множество соотношений, а $v(1{:}|A|)$ – масив логических значений, C – переменная типа множество.

2. Алгоритмы нижней и верхней аппроксимаций

Рассмотрим представление МНА на псевдоязыке.

Входные данные: N_0 входные условия для U-Y схемы A.

Выходные данные: N_a базис инвариантов в состоянии a программы A.

```
N_a0 := N_0
ToVisit := A/{a_0}
for all a ∈ ToVisit do
    N_a:=∅
end for
while ToVisit ≠ ∅ do
    c := takefrom ToVisit
    if N_c ≠ ∅ then
        N := N_c
        for all (a', y, c) do
            N := N ⋂ ef(N_a', y)
        end for
        if then(N ≠ N_c)
            N_c := N
            ToVisit := ToVisit +{a|forevery(c, y, a)}
        end if
    end if
end while
```

Рассмотрим представление МВА на псевдоязыке:

Входные данные: N_0 входные условия для U-Y-программы A.

Выходные данные: N_a базис инвариантов в состоянии a программы A.

```
N_a0 := N_0
ToVisit.push(a_0)
Visited := {}
while ToVisit ≠ ∅ do
    c := ToVisit.pop()
    Visited := Visited + c
    for all (c, y, a') do
        if Not a' in Visited then
            N_a':=ef(N_a', y)
            ToVisit.push(a'))
        end if
    end for
end while
ToVisit := A/{a_0}
while ToVisit ≠ ∅ do
    c := takefrom ToVisit
    if N_c ≠ ∅ then
        N := N_c
        for all (a', y, c) do
            N := N ⋂ ef(N_a', y)
        end for
        if then(N ≠ N_c)
            N_c := N
            ToVisit := ToVisit +{a|forevery(c, y, a)}
        end if
    end if
end while
```

Семантика оператора take a from *ToVisit* такова, что фиксируется элемент a в мно-

жестве $ToVisit$, который при этом удаляется из указанного множества, а оператор выйти из цикла означает завершение наименьшего цикла, в который он входит. Стратегия выбора элементов из $ToVisit$, вообще говоря, произвольная, но предполагается, что всегда последним элементом, извлекаемым из $ToVisit$, является заключительное состояние. Множество простых путей, по которому определяется начальное приближение, в числе входных параметров не фиксируется.

Наряду с приведенными алгоритмами можно рассматривать их версии, ориентированные на работу с алгебраическими базисами. Они отличаются тем, что значением входного параметра N является не само множество соотношений, а его алгебраический базис P, и вместо операции пересечения $\bigcap$ множеств соотношений следует рассматривать операцию $\bigcap$ построения базиса их пересечения.

Некоторые общие свойства описанных алгоритмов МВА и МНА при некоторых ограничениях вытекают из нижеприведенных утверждений [4, 6]. В силу двойственности отношения частичного порядка достаточно рассмотреть, например, убывающую цепочку множеств соотношений

$$N_1 \supseteq N_2 \supseteq N_3 \supseteq \ldots \supseteq N_n \supseteq \ldots, \qquad (3)$$

имеющую место в некотором состоянии a U - Y -программы A.

Теорема 4. Если свободная алгебра $T_D(R)$ конечно порождена и каждый элемент N_i последовательности (3) является нормальной конгруэнцией, то эта последовательность стабилизируется через конечное число шагов.

Из этой теоремы получаем следующие утверждения.

Следствие 1. Если элементы последовательности (3) обладают алгебраическими базисами и, в частности, конечными алгебраическими базисами, то эта последовательность стабилизируется через конечное число шагов.

Отметим еще некоторые простые свойства алгоритмов генерации инвариантных соотношений, которые вытекают из теоремы 4.

Теорема 5. Если в U - Y -программе множество N_0 и пересечение нормальных конгруэнций есть нормальные конгруэнции, то алгоритм МВА заканчивает свою работу после конечного числа шагов.

Действительно, по условию теоремы алгоритм МВА в каждом состоянии U - Y -программы строит убывающую цепочку нормальных конгруэнций $N_a^{(0)} \supseteq \supseteq N_a^{(1)} \supseteq \ldots \supseteq N_a^{(n)} \supseteq \ldots$, которая конечна в силу теоремы 4.

Теорема 6. Если в U - Y -программе N_0 – нормальная конгруэнция и пересечение нормальных конгруэнций есть нормальная конгруэнция, то алгоритм МНА завершает свою работу после конечного числа шагов.

Действительно, по условию теоремы и согласно теореме 4 алгоритм МНА в каждом состоянии U - Y -программы строит возрастающую цепочку нормальных конгруэнций $N_a(0) \subseteq N_a(1) \subseteq \ldots$ $\ldots \subseteq N_a^{(n)} \subseteq \ldots$, которая конечна в силу выполнения условия обрыва возрастающих цепей.

Приведенные теоремы и их следствия дают теоретическую завершимость алгоритмов генерации множеств инвариантных соотношений, однако в них присутствует некоторый элемент неконструктивности – неизвестна длина последовательности (3). При рассмотрении конкретных алгебр эта длина может уточняться. Однако практическая ценность этих теорем состоит в том, что они показывает, что полнота алгоритмов генерации инвариантных соотношений зависит от точности алгоритмов решения задач о соотношениях и о пересечении. Если эти алгоритмы дают точные решения, то алгоритмы МНА и МВА дают максимально возможное множество инвариантов.

Попытаемся точно сформулировать условия, при выполнении которых алгоритмы генерации инвариантных соотношений дают полные системы соотношений.

Теорема 7. Если множество N_0 и множества соотношений, получаемых в каждом состоянии U-Y-программы в процессе работы алгоритма МВА, нормальные конгруэнции, то результат работы алгоритма не зависит от порядка обхода состояний U-Y-программы (и от выбора начальной совокупности простых путей), а множество инвариантов N_a для всякого состояния $a \in A$ совпадает с множеством

$$\bigcap_{l=l(a_0,a)} ef(N_0, y_l).$$

Доказательство этой теоремы можно найти в [4, 6], позволяющее ввести такое определение.

Определение 1. Множество инвариантов любого состояния U-Y-программы совпадающее с множеством $\bigcap_{l=l(a_0,a)} ef(N_0, y_l)$, называется полным относительно генератора Gen, языка инвариантов L и начального множества N_0.

Теорема 7 дает условия, при которых алгоритм МВА генерирует полные множества инвариантов относительно начальной совокупности соотношений. Когда функция ef не является дистрибутивной, получение полной системы соотношений становится неразрешимой задачей, что вытекает из следующего утверждения.

Теорема 8. Если функция ef не является дистрибутивной, то можно указать U-Y-программу А с начальным множеством соотношений N_0, для которой не существует алгоритма, генерирующего полную систему инвариантов относительно N_0 для этой U-Y-программы.

Доказательство этой теоремы получается путем сведения данной проблемы к модифицированной проблеме соответствий Поста [3].

Учитывая данную теорему возникает естественный вопрос, а на какое множество инвариантов можно рассчитывать в случае монотонной функции ef и какие свойства этих множеств? Ответ на этот вопрос дает следующая теорема.

Теорема 9. Если алгоритм МВА завершает свою работу после конечного числа шагов и функция ef является монотонной, то полученное в результате множество инвариантов N_a для любого состояния $a \in A$ является максимальной фиксированной точкой решения системы уравнений

$$\begin{cases} X_a = X_a \bigcap\limits_{(a',u,y,a)\in S} ef(X_{a'}, y) \\ X_{a_0} = N_0 \end{cases}.$$

На завершение приведем некоторые экспериментальные данные, полученные в результате реализации алгоритмов МВА и МНА.

3. Экспериментальные результаты реализации

Эффективность применения алгоритмов МВА и МНА зависит от уровня представления программы, поскольку множества инвариантных соотношений становятся богаче с понижением уровня представления. Иллюстрацией этого высказывания является хорошо известная фортрановская программа умножения квадратных матриц размерности $n \times n$ – $MULT(X, Y, n)$:

```
MULT(A, B, n)
X, Y, Z : array[1 : n]
i, j, k, n : integer
for i = 1 to n do
    for j = 1 to n do
        c(i, j) := 0
        for k = 1 to n do
            z(i, j) := z(i, j) + x(i, k)y(k, j)
        end for
    end for
end for
```

На этом уровне алгоритмы МНА и МВА не дают никаких соотношений, но если применить эти алгоритмы к представлению этой же программы на промежуточном уровне, тогда U-Y программа имеет вид представленный на рис. 1.

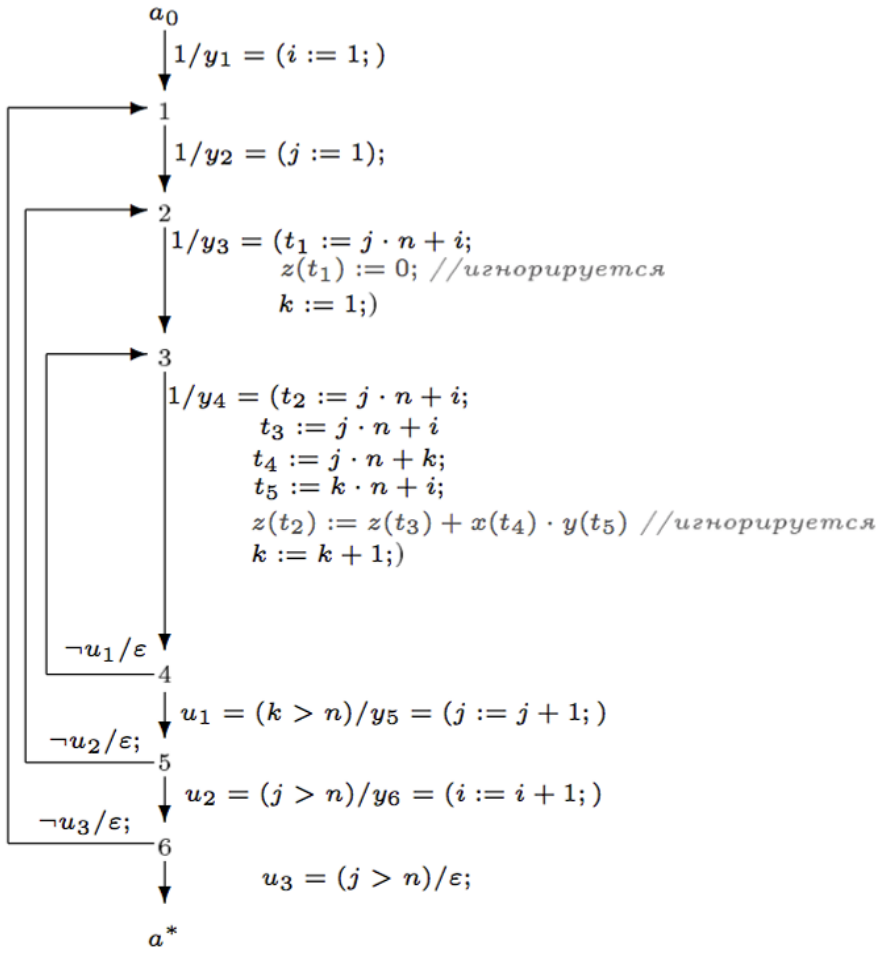

Рис. 1. U - Y -программа MULT(X, Y, n)

Этот уровень представления программы получается учитывая то, что элементы исходных матриц A и B, находящиеся в двумерных массивах, преобразуются для размещения в одномерном массиве Z. Это преобразование выполняет компилятор языка Фортран. Приведенная выше U - Y -программа (рис. 1), алгебра данных которой считается абсолютно свободной, записанная в более общем языке, чем язык стандартных U - Y -программ, в связи с наличием операторов манипуляции с массивами. Эти операторы игнорируются при работе алгоритма.

Алгоритм МВА, будучи примененным к этой U - Y -программе, алгебра данных которой предполагается абсолютно свободной, генерирует следующие инварианты для состояний программы:

$$N_{a_0} = \varnothing,$$

$$N_{a_1} = \varnothing,$$

$$N_{a_2} = \varnothing,$$

$$N_{a_3} = \{t_1 = j * n + i\},$$

$$N_{a_4} = \{t_1 = j * n + i, t_2 = t_1, t_3 = t_1\},$$

$$N_{a_5} = \{t_2 = t_1, t_3 = t_1\},$$

$$N_{a_6} = \{t_2 = t_1, t_3 = t_1\},$$

$$N_{a^*} = N_6 = \{t_2 = t_1, t_3 = t_1\}.$$

Анализируя инварианты мы можем оптимизировать использование переменных, имеющих одинаковые значения t_1, t_2, t_3. Так в состоянии a_3 мы можем оптимизировать одной переменной вместо 3-х. Оптимизированная программа имеет вид (рис. 2).

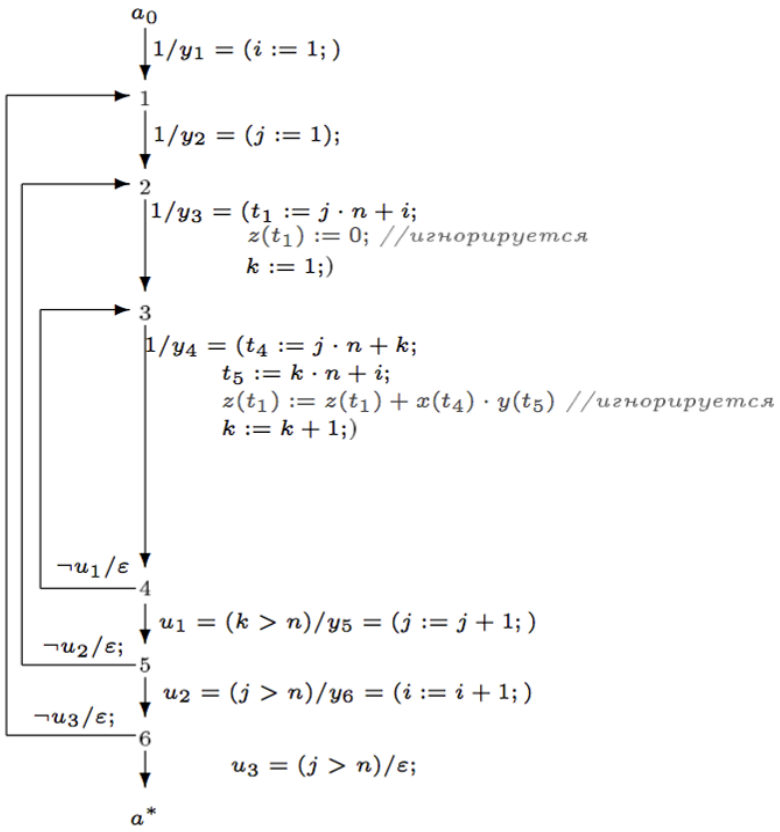

Рис. 2. Оптимизированная U - Y -программа MULT(X, Y, n)

Рассмотрим U - Y -программу из рис. 3, для удобства приведенную далее. На этой программе продемонстрируем роботу алгоритма считая алгебру данных кольцом полиномов [7]. Применяя алгоритм МВА к этой U - Y -программе получаем далее приведенную цепочку вычислений и инвариантные соотношения.

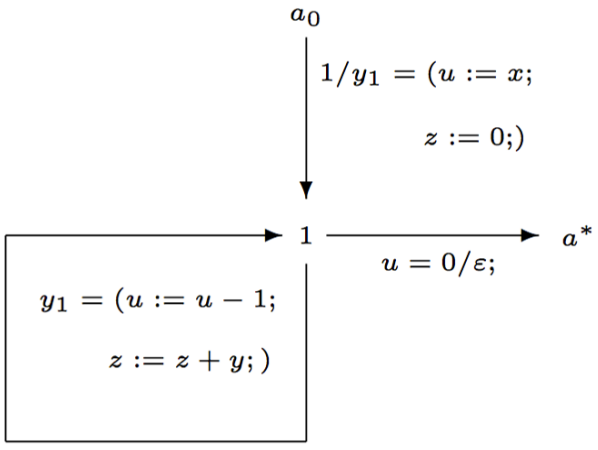

Рис. 3. U - Y -программа УМН(x, y)

На первом этапе алгоритма МВА генерируем начальные множества соотношений. Сначала множествам соотношений для всех состояний программы присваивается значение 1 ($\forall P : P \bigcap 1 = P$), при этом $ToVisit = \{a_0\}$.

Далее рассматривается состояние a_0 из $ToVisit$, фиксируя $Visited = \{a_0\}$. Из состояния a_0 есть один переход $(a_0, 1, y = (u := x; z := 0), a_1)$.

Анализируя переход данного множества соотношений P_{a_1} присваивается:

$$P_{a_1} := ef(P_{a_0}, y) = ef(1, (u := x, z := 0)) =$$
$$= u - x = 0, z = 0.$$

При этом $ToVisit = \{a_1\}$.

Переходим к состоянию a_1 из $ToVisit$, фиксируя $Visited = \{a_0, a_1\}$. Из состояния a_1 есть два перехода $(a_1, (u \neq 0), (u := u - 1; z := z + y), a_1)$, $(a_1, (u = 0), \varepsilon, a^*)$. Исходя из $a_1 \in Visited$ рассмотрению под-

лежит только переход из a_1 в a^*.

Анализируя переход $(a_1, (u = 0), \varepsilon, a^*)$ находим целесообразным заменить этот переход следующим переходом $(a_1, 1, u := 0, a^*)$. То есть заменить объязательное условие перехода оператором присваивания. Множеству соотношений P_{a^*} присваивается:

$$P_{a^*} := ef(P_{a_1}, y) =$$
$$= ef(\{u - x = 0, z = 0\}, \{u := 0\}) =$$
$$= \{x = 0, z = 0\}.$$

При этом $ToVisit = \{a^*\}$.

Переходим к состоянию a^* из $ToVisit$, фиксируя $Visited = \{a_0, a_1, a^*\}$. Нет переходов из состояния a^*. Исходя из $ToVisit = \varnothing$ первый этап МВА закончен. Переходим ко второму этапу алгоритма.

На втором этапе алгоритма на каждом шаге итерации генерируем соотношения для состояний. После пересекаем множества соотношений с полученными на предыдущем шаге. При инициализации $ToVisit = A/\{a_0\} = \{a_1, a^*\}$.

Переходим к состоянию a_1 из $ToVisit$ $P = P_{a_1} = \{u - x = 0, z = 0\}$. В состоянии a_1 есть два перехода:

$$(a_0, 1, (u := x; z := 0), a_1),$$

$$(a_1, (u \neq 0), (u := u - 1; z := z + y), a_1).$$

Анализируя переход $(a_0, 1, (u := x; z := 0), a_1)$, множеству соотношений P присваивается:

$$P := P \bigcap ef(P_{a_0}, y) :=$$
$$:= P \bigcap ef(1, (u := x; z := 0)).$$

Анализируя переход $(a_1, (u \neq 0), (u := u - 1; z := z + y), a_1)$, множеству соотношений P присваивается:

$$P := P \bigcap ef(P_{a_1}, y) :=$$
$$:= P \bigcap ef(\{u - x = 0, z = 0\}, (u := u - 1; z :=$$
$$:= z + y)) = \{u - x = 0, z = 0\} \bigcap \{u + 1 - x =$$
$$= 0, z - y = 0\} = \{z + (u - x)y = 0,\ z(y - z) =$$
$$= 0,\ zx - zu - z = 0,\ (u - x)^2 + u - x = 0\}.$$

Исходя из $P \neq P_{a_1}$ присваиваем

$$P_{a_1} := \{z + (u - x)y = 0, z(y - z) = 0,$$

и

$$zx - zu - z = 0, (u - x)^2 + u - x = 0\},$$

$$ToVisit := ToVisit \bigcup \{a_1, a^*\} = \{a_1, a^*\}.$$

Переходим к состоянию a^* из $ToVisit$, фиксируя $Visited = \{a_1\}$ и $P = P_{a^*} = \{x = 0, z = 0\}$. В состоянии a^* есть один переход $(a_1, 1, u := 0, a^*)$.

Анализируя переход $(a_1, 1, u := := 0, a^*)$, множеству соотношений P присваивается.

Исходя из $P \neq P_{a^*}$ присваиваем

$$P_{a^*} := \{z(x - 1) = 0, z(y - z) = 0,$$
$$(x^2 - x)u = 0, (xy - z)u = 0\}.$$

Переходим к состоянию a_1 из $ToVisit$, фиксируя $Visited = \varnothing$ и

$$P = P_{a_1} = \{z + (u - x)y = 0, z(y - z) = 0,$$
$$zx - zu - z = 0, (u - x)^2 + u - x = 0\}.$$

В состоянии a_1 есть два перехода:

$$(a_0, 1, (u := x; z := 0), a_1),$$

$$(a_1, (u \neq 0), (u := u - 1; z := z + y), a_1).$$

Анализируя переход $(a_0, 1, (u := x; z := 0), a_1)$, множеству соотношений P присваивается:

$$P := P \bigcap ef(P_{a_0}, y) := P \bigcap \{u - x = 0, z = 0\} =$$
$$= \{z + (u - x)y = 0, z(y - z) = 0, zx - zu - z =$$
$$= 0, (u - x)^2 + u - x = 0\} \bigcap \{u - x = 0, z = 0\} =$$
$$= \{(x - u)y - z = 0, zy - z^2 = 0, zx - zu - z =$$
$$= 0, (u - x)^2 + u - x = 0\}.$$

Анализируя переход $(a_1,(u \neq 0),(u := := u-1; z := z+y), a_1)$, множеству соотношений P присваивается:

$$P := P\bigcap ef(P_{a_1}, y) := P\bigcap ef(P_{a_1},(u := u-1;$$
$$z := z+y)) = \{xy-uy-z = 0,\ldots\}.$$

Исходя из $P \neq P_{a_1}$ присваиваем

$$P_{a_1} := \{xy-uy-z = 0,\ldots\}$$

и

$$ToVisit := ToVisit \bigcup \{a_1, a^*\} = \{a_1, a^*\}.$$

Переходим к состоянию a^* из $ToVisit$, фиксируя $Visited = \{a_1\}$ и

$$P = P_{a^*} = \{z(x-1) = 0, z(y-z) = 0,$$
$$(x^2-x)u = 0, (xy-z)u = 0\}.$$

В состоянии a^* есть один переход $(a_1, 1, u := 0, a^*)$.

Анализируя этот переход множеству соотношений P присваивается:

$$P := P\bigcap ef(P_{a_1}, u := 0) := P\bigcap \{xy-z = 0,\ldots\} = \{xy-z = 0,\ldots\}.$$

Исходя из $P \neq P_{a^*}$ присваиваем $P_{a^*} := := \{xy-z = 0,\ldots\}$.

Выполняя действия по алгоритму мы заметили, что с каждой новой итерацией размерность базисов P_{a_1} и P_{a^*} остается неизменной, также как и один из многочленов этих базисов. Это многочлен $xy-uy-z = 0$ в a_1 и $xy-z = 0$ в a_0. Как мы видим из постановки задачи инвариант $z = xy$ подтверждает правильность программы. В этом примере работа автоматического доказчика не понадобилась.

Рассмотрим еще один пример программы которая вычисляет

$$\sum_{y=1}^{n} \frac{\sum_{z=1}^{m-1} z^y}{m^y}$$

по входным параметрам m и n. На высоком уровне U-Y программы показана на рис. 4.

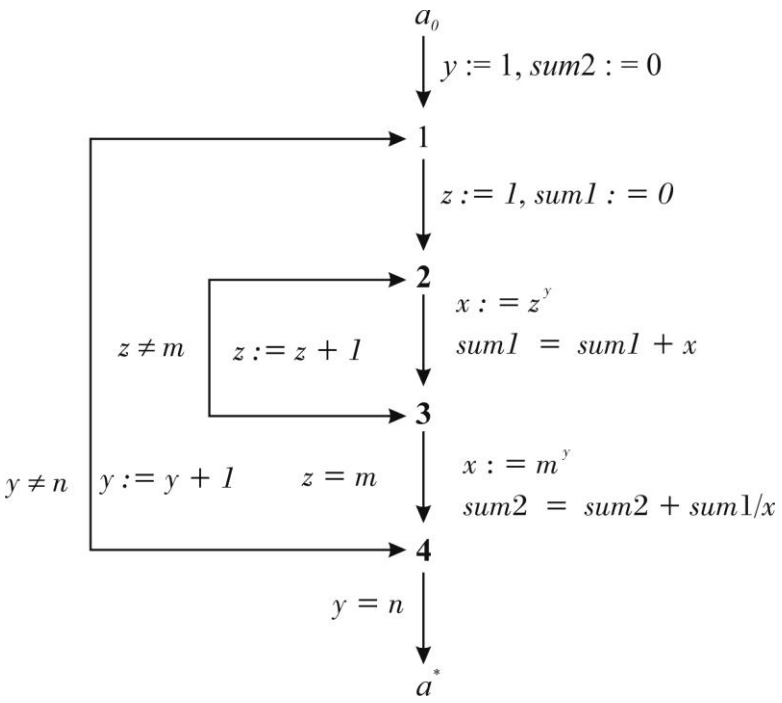

Рис. 4. U-Y программа $\sum_{y=1}^{n} \frac{\sum_{z=1}^{m-1} z^y}{m^y}$

Рассмотрим роботу алгоритма МВА для программ над абсолютно свободными алгебрами данных.

Инициализируем $ToVisit := \{a_0\}$. Переходим к состоянию a_0 из $ToVisit$, фиксируя $Visited = \{a_0\}$. Анализируем переход $(a_0, (sum2 := 0, y := 1), a_1)$, множеству соотношений P_{a_1} присваивается:

$$P_{a_1} := ef(P_{a_0}, (sum2 := 0, y := 1)) =$$

$$= \{sum2 = 0, y = 1\},$$

ведь $P_{a_0} = \varnothing$.

При этом $ToVisit = \{a_1\}$.

Переходим к состоянию a_1 из $ToVisit$, фиксируя $Visited = \{a_0, a_1\}$. Анализируем переход $(a_1, (sum1 := 0, z := 1), a_2)$, множеству соотношений P_{a_1} присваивается:

$$P_{a_2} := ef(P_{a_1}, (sum1 := 0; z := 1)) =$$

$$= ef(\{sum2 = 0, y = 1\}, (sum1 := 0; z := 1)) =$$

$$= \{sum2 = 0, sum1 = sum2, z = 1, y = z\}.$$

При этом $ToVisit = \{a_2\}$.

Переходим к состоянию a_2 из $ToVisit$, фиксируя $Visited = \{a_0, a_1, a_2\}$. Продолжая выполнение первого этапа МВА, в результате получаем следующие базисы соотношений:

$P_{a_0} = \varnothing,$

$P_{a_1} = \{sum2 = 0, y = 1\},$

$P_{a_2} = \{sum2 = 0, sum1 = sum2, z = 1, y = z\},$

$P_{a_3} = \{sum2 = 0, sum1 = sum2 + x, z = 1,$

$y = z, x = z^z\},$

$P_{a_4} = \{sum2 = 0 + sum1, sum1 = 0 + z^z, z = m,$

$y = 1, x = z^y\},$

$P_{a^*} = \{sum2 = 0 + 0 + z^z, sum1 = 0 + z^z, z = m,$

$y = n, x = z^1\}.$

Рассмотрим выполнение второго этапа алгоритма. При инициализации $ToVisit = \{a_1, a_2, a_3, a_4, a^*\}$.

Переходим к состоянию a_1 из $ToVisit$ $P = P_{a_1} = \{u - x = 0, z = 0\}$. В состоянии a_1 есть два перехода $(a_0, (sum2 := := 0, y := 1), a_1)$, $(a_4, (y := y + 1), a_1)$.

Анализируя переход $(a_0, (sum2 := := 0, y := 1), a_1)$, множеству соотношений P присваивается:

$$P := P \bigcap ef(P_{a_0}, y) = P \bigcap ef(\varnothing, (sum2 := 0,$$

$$y := 1)) = \{sum2 = 0, y = 1\}.$$

Анализируя переход $(a_4, (y := y + + 1), a_1)$, множеству соотношений P присваивается:

$$P := P \bigcap ef(P_{a_4}, y) := P \bigcap ef(\{sum2 =$$

$$= 0 + sum1, sum1 = 0 + z^z, z = m, y = 1,$$

$$x = z^y\}, (y := y + 1)) = \{u - x = 0, z = 0\} \bigcap$$

$$\bigcap \{u + 1 - x = 0, z - y = 0\} = \{(x - u)y - z =$$

$$= 0, zy - z^2 = 0, zx - zu - z = 0,\ u^2 + u - 2xu -$$

$$- 2xu - x + x^2 = 0\} = \{sum2 = 0 + 0 + z^z,$$

$$sum1 = 0 + z^z, z = m, y = 1 + 1, x = z^1\}.$$

Исходя из $P \neq P_{a_1}$ присваиваем

$$P_{a_1} := \{sum2 = 0, y = 1\} \bigcap \{sum2 = 0 + 0 + z^z,$$

$$sum1 = 0 + z^z, z = m, y = 1 + 1, x = z^1\} = \varnothing$$

и $ToVisit := ToVisit \bigcup \{a_2\} = \{a_2, a_3, a_4, a^*\}$.

Переходим к состоянию a_2 из

$ToVisit$ $P = P_{a_2} = \{sum2 = 0, sum1 = sum2,$

$$z = 1, y = z\}.$$

В состоянии a_1 есть два перехода

$(a_1, (sum1 := 0, z := 1), a_2)$ $(a_3, (z := z + 1), a_2)$.

Анализируя переход $(a_1, (sum1 := 0,$ $z := 1), a_2)$, множеству соотношений P присваивается:

$$P := P\bigcap ef(P_{a_1}, (sum1 := 0, z := 1)) =$$

$$= P\bigcap ef(\varnothing, (sum1 := 0, z := 1)) =$$

$$= \{sum1 = 0, z = 1\}.$$

Анализируя переход $(a_3, (z := z+1), a_2)$, множеству соотношений P присваивается:

$$P := P\bigcap ef(P_{a_3}, y) := P\bigcap ef(\{sum2 = 0,$$

$$sum1 = sum2 + x, z = 1, y = z, x = z^z\},$$

$$(z := z+1)) = \{sum2 = 0, sum1 = sum2, z = 1,$$

$$y = z\}\bigcap\{sum2 = 0,\ sum1 = sum2 + z^z,$$

$$z = y + y, y = 1, x = y^y\} = \varnothing.$$

Исходя из $P \neq P_{a_2}$ присваиваем

$$P_{a_2} := \varnothing$$

и $ToVisit := ToVisit\bigcup\{a_3\} = \{a_3, a_4, a^*\}$.

Переходим к состоянию a_3 из

$$ToVisit\ P = P_{a_3} = \{sum2 = 0, sum1 =$$

$$= sum2 + x, z = 1, y = z, x = z^z\}.$$

В состоянии a_1 есть один переход $(a_2, (sum1 := sum1 + z^y, z := z^y), a_3)$.

Анализируя переход $(a_2, (sum1 :=$ $:= sum1 + z^y, z := z^y), a_3)$, множеству соотношений P присваивается:

$$P := P\bigcap ef(P_{a_2}, y) = \{sum2 = 0,$$

$$sum1 = sum2 + x,\ z = 1, y = z,$$

$$x = z^z\}\bigcap ef(\varnothing, (sum1 := sum1 + z^y,$$

$$z := z^y)) = \{x = z^y\}.$$

Исходя из $P \neq P_{a_3}$ присваиваем

$$P_{a_3} := \{x = z^y\}$$

и

$$ToVisit := ToVisit\bigcup\{a_2, a_4\} = \{a_2, a_4, a^*\}.$$

Переходим к состоянию a_2 из $ToVisit$. Исходя из того, что $P_{a_2} = \varnothing$ не рассматриваем это состояние.

Переходим к состоянию a_4 из

$$ToVisit\ P = P_{a_4} = \{sum2 = 0 + sum1, sum1 =$$

$$= 0 + z^z, z = m, y = 1, x = z^y\}.$$

В состоянии a_1 есть один переход

$$(a_3, (sum2 := sum2 + sum1/x, x := m^y), a_4).$$

Анализируя переход

$$(a_3, (sum2 := sum2 + sum1/x, x := m^y,$$

$$z := m), a_4),$$

множеству соотношений P присваивается:

$$P := P\bigcap ef(P_{a_3}, y) = \{sum2 = 0 + sum1,$$

$$sum1 = 0 + z^z, z = m, y = 1, x = z^y\}\bigcap$$

$$\bigcap ef(\{x = z^y\}, (sum2 := sum2 + sum1/x,$$

$$x := m^y, z := m)) = \{z = m, x = z^y\}.$$

Исходя из неравенства базисов $P \neq P_{a_4}$ присваиваем

$$P_{a_4} := \{z = m, x = z^y\}$$

и

$$ToVisit := ToVisit\bigcup\{a_1, a_5\} = \{a_1, a_4, a^*\}.$$

Переходим к состоянию a_1 из $ToVisit$. Исходя из того, что $P_{a_1} = \varnothing$ не рассматриваем это состояние.

Переходим к состоянию a^* из

$$ToVisit\ P = P_{a^*} = \{sum2 = 0 + 0 + z^z,$$

$$sum1 = 0 + z^z, z = m, y = n, x = z^1\}.$$

В состоянии a^* есть один переход $(a_4, y := n, a^*)$.

Анализируя переход $(a_4, y := n, a^*)$, множеству соотношений P присваивается:

$$P := P \bigcap ef(P_{a_4}, (y := n)) =$$

$$= \{sum2 = 0 + 0 + z^z, sum1 = 0 + z^z, z = m,$$

$$y = n, x = z^1\} \bigcap ef(\{z = m, x = z^y\}, (y := n)) =$$

$$= \{z = m, y = n\}.$$

Исходя из неравенства базисов $P \neq P_{a_*}$ присваиваем $P_{a_*} := \{z = m, y = n\}$. Множество *ToVisit* пустое по этому алгоритм заканчивает свою работу.

Результатом выполнения алгоритма будет соотношения состояниям программы и базисы инвариантов:

$$P_{a_0} = \varnothing,$$

$$P_{a_1} = \varnothing,$$

$$P_{a_2} = \varnothing,$$

$$P_{a_3} = \{x = z^y\},$$

$$P_{a_4} = \{x = z^y, z = m\},$$

$$P_{a_*} = \{y = n, z = m\}.$$

Полученные базисы инвариантов могут являться входными параметрами для автоматического доказателя. Поскольку в состоянии a_3 используется главная операция суммирования $sum1 := sum1 + x$ Таким образом инвариант $x := z^y$ указывает, что сумма считается правильно. В состоянии a_4 выполняется операция деления на x и правильными есть условия $x = z^y$ и $z = m$, т. е. деление выполняется правильно относительно постановки задачи. Инварианты в a^* дают возможность проверить условия выхода из программы. Но в отличие от предыдущего примера программы на низком уровне инварианты в рассмотренном примере слабее.

Результаты реализации. Для реализации алгоритма МВА над абсолютно свободной алгеброй данных использован язык Java и использована библиотека ANTLR для построения представления выражений в виде дерева. Структура классов разделена на представление $U\text{-}Y$ программы и реализацию методов аппроксимации. Представлением $U\text{-}Y$ программы является класс $U\text{-}Y$ программы, состоящий из ориентированного графа. Вершины этого графа – объекты класса состояний $U\text{-}Y$ программы, а ребра – объекты класса переходов в $U\text{-}Y$ программе. Класс переходов в $U\text{-}Y$ программе включает в себя список выражений для рассматриваемой алгебры, реализованных в виде отдельного класса.

Реализации алгоритмов МВА и МНА базируется на абстрактном классе итерационного алгоритма. Этот класс содержит список пар вида состояния $U\text{-}Y$ программы и базис множества соотношений в этом состоянии. Этот же класс содержит три метода: метод выполнения шага алгоритма, метод проверки условия окончания работы алгоритма и метод обхода состояний программы.

Для получения ориентировочных оценок быстродействия алгоритма автоматически генерируется $U\text{-}Y$ схема программы в предположении абсолютно свободной алгебры, имеющей большое количество вершин и переходов. При этом обеспечивается условие непустоты множества инвариантов программы.

Такой программой послужила программа из примера 1. В этой программе добавлено еще одно ветвление от вершины 1. В этом ветлении, в автоматическом режиме, добавлялись новые состояния и копии ребра

$$(a_1, (u \neq 0), (u := u - 1; z := z + y), a_1),$$

составляющие цепочку с одинаковыми операторами присваивания. В этом случае, схема теряла свой алгоритмический смысл, но позволяла оценить бытстродействие алгоритма в зависимости от размера $U\text{-}Y$ схемы.

Целью эксперимента являлось определение зависимости количества элементарных операций в МВА от таких параметров: количества состояний схемы, количества переходов между состояниями, количества переменных и количества операций присваивания в программе. Результаты выполнения эксперимента представлены ниже в таблице.

Таблица. Показатели быстродействия МВА

N	Кол-во состояний	Кол-во переходов	Кол-во переменных	Кол-во присваиваний	Кол-во операций
1.	7	8	3	14	432
2.	31	40	3	70	5564
3.	91	120	3	210	9271
4.	211	280	3	490	24559
5.	6001	8000	3	14000	1500078

Реализация алгоритма МВА для кольца полиномов является более сложной. Принято решение использовать более высокоуровневый, с математической точки зрения, инструмент, это вычислительный пакет Maple. В этом случае для представления достаточно внутренних типов Maple. Функция пересечения двух базисов множеств равенств в силу теоремы [7] реализована с помощью пакета Groebner и является креугольным камнем данной реализации. Реализация алгоритма обхода состояний программы практически является транслированной реализации обхода, использующийся для абсолютно свободных алгебр.

Выводы

В статье приведено доказательство полноты множества генерируемых соотношений инвариантов при условии дистрибутивности операции ef для языка равенств. Показано отсутствие полного множества инвариантных соотношений в случае не дистрибутивности операции ef относительно операции пересечения.

Одним из основных результатов проведенных исследований есть реализация методов МВА и МНА и их экспериментальная апробация. Проиллюстрирована эффективность методов на низком уровне представления программ и их слабая эффективность на высокоуровневом представлении. Приведен пример работы алгоритма МВА для программ, алгебрами данных которых являются кольца полиномов. Алгоритм МВА показал себя более интересным с практической точки зрения.

Современные программные технологии выдвигают все более высокие требования к возможностям верификации программ. Учитывая это, предложенные методы нуждаются в дальнейшем развитии. Проведенные исследования указывают три направления развития: расширение алгебры данных, переход к объектно ориентированному представлению переменных, представления условий в виде языка неравенств. Расширение работы алгоритмов МВА и МНА на более сложные алгебры данных позволит анализировать более сложные формулы в операторах присваивания. На практике наиболее используемыми являются условия в виде неравенств, поэтому инварианты будут более точными, если учесть эти дополнительные условия на ветвлениях U-Y схем программ.

Проведение таких исследований дает возможность вплотную приблизится к верификации сложных программ, которые используют объектно ориентированные языки программирования на высоком уровне. Решение этих задач может найти применения в таких верификационных комплексах как JPF (http://babelfish.arc.nasa.gov/).

1. *Максимец А.Н.* Верификация программ: состояние, проблемы, экспериментальные результаты I // Проблеми програмування. – 2013. – № 4. – С. 53–63.

2. *Летичевский А.А.* Эквивалентность и оптимизация программ // Труды междунар. симпозиума по теоретическому программированию. – Новосибирск. – 1972. – С. 166–180.
3. *Kam J.B., Ullman D.J.* Monotone data flow analysis frame works // Acta Inform. – 1978. – N 3. – P. 305–318.
4. *Годлевский А.Б., Капитонова Ю.В., Кривой С.Л., Летичевский А.А.* Итеративные методы анализа программ // Кибернетика. – 1989. – № 2. – С. 9–19.
5. *Летичевский А.А.* Об одном подходе к анализу программ // Кибернетика. – 1979. – № 6. – С. 1–8.
6. *Годлевский А.Б., Капитонова Ю.В., Кривой С.Л., Летичевский А.А.* Итеративные методы анализа программ. Равенства и неравенства // Кибернетика. – 1990. – № 3. – С. 1–9.
7. *Maksymets O.M.* Upper approximation method for polynomial invariants Theoretical and Applied Aspects of Cybernetics // Proceedings of the 2nd International Scientific Conference of Students and Young Scientists. Computer Science Section. – Kyiv: Bukrek. – 2012. – P. 45–47.

Получено 07.03.2013

Об авторе:

Максимец Александр Николаевич,
аспирант.

Место работы автора:

Киевский национальный университет
имени Тараса Шевченко,
факультет кибернетики.
Тел: +38 050 9852274.
E-mail: maksymets@gmail.com