УДК 004.056.57

В.В. Ковалев, Р. И. Компаниец, В.А. Новиков

ВЕРИФИКАЦИЯ ПРОГРАММ НА ОСНОВЕ СООТНОШЕНИЙ ПОДОБИЯ

Ковалев В.В., Компаниец Р.И., Новиков В.А. **Верификация программ на основе соотношений подобия.**

Аннотация. Описывается подход к статической верификации исполняемых программ на основе сопоставления семантических аспектов вычислений, позволяющий построить паспорт программы. Паспорт, как результат статической верификации, может быть использован для создания среды контролируемого выполнения программ (динамической верификации реально выполняемых программ, прошедших статическую верификацию).

Ключевые слова: верификация, управляющий граф, абстрактные размерности, определяющие отношения, подобие.

Kovalev V.V., Kompanietc R. I., Novikov V.A. **Verification of Programs Based on Similarity Relations.**

Abstract. The static verification method of programs based on comparison of the semantic aspects of computing, that allows to build a program passport is described. Passport as a result of the static verification may be used to create an environment of the controlled program run (dynamic verification of the really running programs, which have passed static verification).

Keywords: verification, control graph, abstract dimensions, determinant relations, similarity.

1. Введение. Целью верификации программ, в отличие от отладки, является выявление ошибок и условий их возникновения. Кроме того, верификация - контролируемый и управляемый процесс, трудоемкость которого определяется используемым методом с предопределенным в нем объемом работ. Предлагаемый метод для получения абстрактной интерпретации состояний программы использует положения теории размерностей, а для их анализа на непротиворечивость – положения теории подобия. Получаемые в процессе символьной интерпретации критерии подобия объединяются в систему линейных однородных уравнений (СЛОУ), которая и ее решение служат паспортом программы. Опираясь на паспорт можно с помощью сопроцессора контролировать семантическую правильность (динамически верифицировать) выполнения программы, прошедшей статическую верификацию.

2. Статическая верификация исполняемых программ. Верификация осуществляется на основе анализа однозначности функциональности ветвей в пределах артикулирующих компонентов программ (АКП), т. е. АКП рассматриваются как элементы факторизации управляющего графа (УГ) программы (рисунок 1) при выявлении недекларированного выполнения (НДВ) программы на основе, например, закладки.

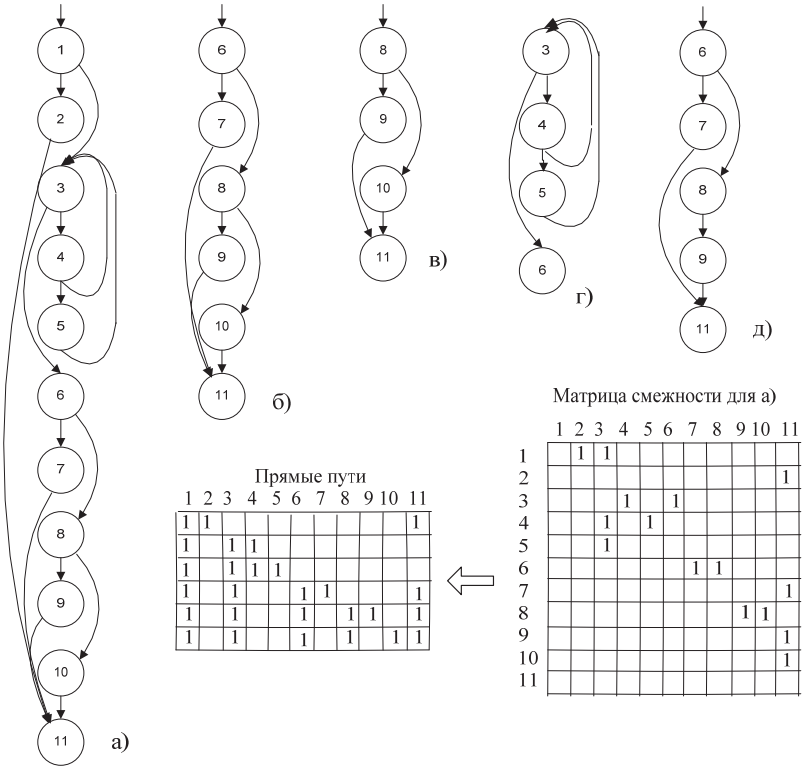

Рис. 1. Управляющий граф анализируемой подпрограммы (а) и возможные артикулирующие компоненты программ (б, в, г, д)

УГ на рисунке 1 соответствует программе, исходный текст которой приведен в листинге 1.

```
int _tmain(int argc, _TCHAR* argv[])
{
        string passwd_file = "password.txt";
        ifstream passwd_stream( passwd_file );
        if ( ! passwd_stream.is_open() )//1
        {
                string err = strerror( errno );
                cerr << "Cannot open file " << passwd_file << " : "
                 << errno << " : " << err << endl;
                exit( -1 );//2
        }
        string line;
        while( getline( passwd_stream, line ) ) {.//3
```

```
                istringstream iss(line);
                string name;
                string password;
                if ( !(iss >> name >> password) ) continue;//4
                passwords[name] = password;//5
        }
        string name, password;//6
        cout << "Enter name:"; cin >> name;//6
        cout << "Enter password:"; cin >> password;//6
        if ( name == "admin" && password == "111" ){//6
                cout << "Welcome!" << endl; //7
        }
        else{
                 if (passwords.find(name)!= passwords.end()
                 &&passwords[name]==password) //8
                         cout << "Welcome!" << endl;//9
                else     cout << "Access denied!" << endl;//10
        }
        return 0;//11
}
```

Листинг 1. Исходный текст анализируемой программы

Под АКП будем понимать как УГ отдельной подпрограммы, так и УГ фрагментов программ, ограниченных артикуляционными (субартикуляционными для вложенных фрагментов) вершинами. На рисунке 1 представлен УГ исследуемой программа и возможные АКП для неё, т.е. АКП – фрагмент УГ, ограниченный вершинами одинакового уровня вложенности. Постулируем, что в программе существует условие, выполнение которого инициирует закладку. В общем случае фрагмент программы с закладкой должен содержаться в структурах, подобных управляющим структурам, представленным на рисунке 2.

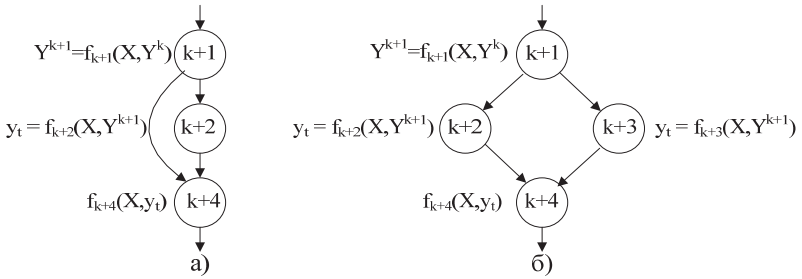

Рис. 2. Элементарные структуры-факторы

Предполагается, что проверяемые программы синтаксически правильные и описывают замкнутый вычислительный процесс Y = F (X, Y), где: X ={x_i} – множество имен исходных данных, а Y ={y_j} – множество имен результатов, включая промежуточные результаты.

Выполним формальный анализ представленных на рисунке 2 элементарных АКП (содержат точно два альтернативных пути), используя технику ограниченных символьных вычислений, положения теорий размерностей и подобия для чего предположим, что у нас имеется отображение T : X $\rightarrow$[X], которое ставит в соответствие каждому $x_i \in$ X некоторую абстрактную сущность или размерность [x_i].

Ограниченные символьные вычисления заключаются в подстановке во все содержащиеся в программе выражения E_i, вместо $y_j \in$ Y определяющих их термов, состоящих из $x_i \in$ X . Если E не содержит $y_j \in$ Y, то оно считается вычисленным или термом. Мы переносим аналитические связи между переменными и константами программы на их абстрактные сущности, т. е. действия выполняются над размерностями, что обеспечивает сохранность семантических аспектов вычислений, а выполнение подстановки делает эти выражения инвариантными к перемещениям, что позволяет объединять их в систему для совместного анализа свойств на непротиворечивость.

Любое, в том числе символьное, выполнение программы осуществляется с учетом управляющей структуры. Тогда в нашем примере на рисунке 2, семантика вычисленного объекта y_t должна быть подобной для разных путей вычисления при входе в вершину b_{k+4}, где эти пути сходятся в пределах АКП. Для примеров на рисунке 2а) и 2б) можно записать следующие равенства, называемые в дальнейшем определяющими отношениями (ОО):

$$\text{для а): } [f_{k+4}(X, f_{k+1}(X, Y_k))] = [f_{k+4}(X, f_{k+2}(X, f_{k+1}(X, Y_k)))], \qquad (1)$$

$$\text{для б) : } [f_{k+4}(X, f_{k+2}(X, Y_{k+1}))] = [f_{k+4}(X, f_{k+3}(X, Y_{k+1}))]. \qquad (2)$$

В предлагаемом подходе нас будет интересовать не численное равенство правой и левой частей в (1) и (2), а их семантическое равенство или подобие как однородных по размерностям величин, т. е. имеющих одинаковые, хотя и абстрактные, размерности. Например, в известных уравнениях:

$$\begin{aligned} a &= (v_t - v_0)/t, \\ s &= v_0 t + at^2/2 \end{aligned} \qquad (3)$$

слагаемые должны иметь одинаковые физические размерности.

Выполнив подстановку *a*, получим:

$$s = v_0t + ((v_t - v_0)/t)t^2/2 = v_0t + v_t t - v_0t. \quad (4)$$

Из (4) мы получим единственное ОО:

$$[v_0t] = [v_t t] \text{ или } [v_0] = [v_t]. \quad (5)$$

Подобную, но созданную управляющими связями, ситуацию и описывают ОО (1) и (2). Для сравнения семантических форм в ОО, подобных (1), (2) и (5), будем использовать технику преобразований теории подобия, для чего ОО делением левой части на правую часть представляются в виде степенных одночленов:

$$\prod_{K=1}^{\Upsilon}[X_k]^{a_k} = 1 \text{ или } \prod_{K=1}^{\Upsilon}[X_k]^{a_k} = [.]^0.$$

Например, используя данную технику преобразований, (4) примет вид:

$$[v_0]\,[v_t]^{-1} = [.]^0. \quad (6)$$

В терминах теории подобия безразмерное соотношение (6) называют критерием подобия и используют при анализе (моделировании) подобных физических явлений, а применительно к нашей ситуации – для проверки семантического подобия альтернирующих вычислений в программах.

Для проверки тождественности критериев подобия, выявленные в АКП степенные одночлены (5) подвергаются последовательно логарифмированию:

$$\sum_{k=1}^{\Upsilon} a_k \ln[x_k] = 0,$$

и после замены переменных ln[xk]=zk, k=1,…,r, рассматриваются как:

$$\sum_{k=1}^{\Upsilon} a_k z_k = 0$$

- линейные однородные уравнения. В пределах АКП таких уравнений может быть несколько и, так как они имеют одинаковое пространство столбцов Z и инвариантны к перемещениям, то в пределах элементарного АКП их можно объединить в систему линейных однородных уравнений (ЛОУ) Az = 0 и проверить на совместность. Если абстрактные размерности назначены n переменным из *X*, то есть $|X| = n$, то нельзя получить более *n-1* ЛОУ. Более того, если ОО выявили, что

только k из n переменных являются независимыми по размерности, то мы можем получить систему лишь из n-k ЛОУ [2]. В нашем случае κ=2. Независимые размерности имеют v и t и можно построить только одно ОО.

Смоделируем ошибку, изменив первое уравнение в (3) на a = $(v_t - v_0)t$, (операция деления заменена умножением). После подстановки a ($a \in Y$) в $s = v_0t + at^2/2$ получим:

$$s = v_0t + ((v_t - v_0)t)t^2/2 = v_0t + v_tt^3 + v_0t^3$$

Для полученного выражения построим три ОО, при этом получим несовместную систему ЛОУ, так как порядок системы ЛОУ не меньше |X|=3, а вычитание второго ЛОУ из первого порождает третье (и наоборот, его сложение с третьим порождает первое). Кроме того [t] становится безразмерной величиной (что невозможно) – сказывается введенная нами ошибка (при манипуляциях с t) и она будет обнаружена экспертом по этим данным.

$$\begin{array}{lll} [v_0t] = [v_tt^3] & [v_0] & [t]^{-2} = [.]^0 \\ [v_0t] = [v_0t^3] \longrightarrow & & [t]^{-2} = [.]^0 \\ [v_0t^3] = [v_tt^3] & [v_0]\,[v_t]^{-1} & = [.]^0 \end{array} \xrightarrow[[Y]=([a],[s])]{[X]=([v_0],[v_t],[t])} A=\begin{pmatrix} 1 & -1 & -2 \\ 0 & 0 & -2 \\ 1 & -1 & 0 \end{pmatrix}.$$

Вы можете сказать, что в нашем примере независимыми размерностями являются [s] и [t]. Это невозможно, так как $s \in Y$, а размерности мы назначаем объектам из X.

АКП могут быть вложенными. Пример анализа вложенных АКП показан на рисунке 3. После проверки соотношения $[f_9(X,Y_8)] = [f_{10}(X,Y_8)]$ и его правильности граф а) может быть редуцирован к б), а последний на основании:

$$[f_7(X,Y_6)] = \begin{Bmatrix} [f_8(X,Y_6)] \\ [f_9(X,Y_8)] \end{Bmatrix}$$

к в) без потери информации о семантических аспектах вычислений, т.е. «правильная программа» должна редуцироваться к единственной вершине.

Рассмотрим пример. Программа, представленная в листинге 1, схематично соответствует процедуре идентификации пользователя. В процедуре есть закладка, реализованная с помощью задания констант во фразе if: «if (name == “admin” && password == “1111”)», позволяющая при необходимости обойти идентификацию пользователя.

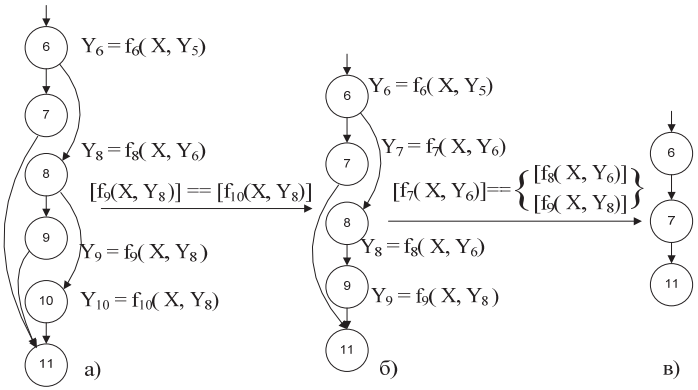

Рис. 3. Анализ (редукция) вложенных АКП

Рисунок 4 фрагментарно демонстрирует результаты анализа программы листинга 1. В общем случае предлагаемый метод должен проверять семантическую тождественность вычислений для каждого представительного АКП, а также при выполнении операций {+, -, :=, ввода-вывода, отношения, с символами и строками}. Здесь мы фиксируем выявленные представительные преобразования, которые отображаем на векторы X и Y и, выполнив подстановку, получим в итоге два ОО.

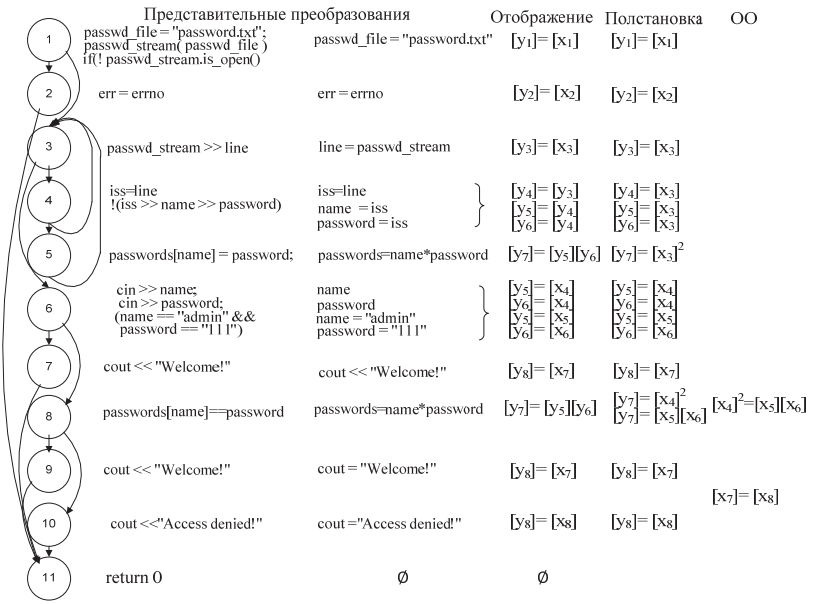

Рис. 4. Этапы анализа программы на НДВ

Процесс анализа разбит на шаги и отображается в соответствующих колонках рисунке 4.

Допустим, что мы анализируем АКП, образованный вершинами 6 – 11 (см. рисунок 4). В АКП(6-11) три пути: (6, 7, 11); (6, 8, 9, 11); и (6, 8, 10, 11). Их представляют подстановки, показанные на рисунке 5.

$$(6,7,11):\left\{\begin{array}{l}[y_5]=[x_4];\\ [y_6]=[x_4];\\ [y_5]=[x_5];\\ [y_6]=[x_6];\\ [y_8]=[x_7].\end{array}\right\}\quad (6,8,9,11):\left\{\begin{array}{l}[y_5]=[x_4];\\ [y_6]=[x_4];\\ [y_5]=[x_5];\\ [y_6]=[x_6];\\ [y_7]=[x_4]\ [x_4];\\ [y_7]=[x_5]\ [x_6];\\ [y_8]=[x_7].\end{array}\right\}\quad (6,8,10,11):\left\{\begin{array}{l}[y_5]=[x_4];\\ [y_6]=[x_4];\\ [y_5]=[x_5];\\ [y_6]=[x_6];\\ [y_7]=[x_4]\ [x_4];\\ [y_7]=[x_5]\ [x_6];\\ [y_8]=[x_8].\end{array}\right\}$$

Рис. 5. Пути в АКП(6 - 11) после выполнения подстановки

Анализ АКП(6-11) начнется с анализа вложенного в него элементарного АКП(8-11), которому соответствуют пути (подстановки – символьного выполнения) (6, 8, 9, 11) и (6, 8, 10, 11). Альтернирующие пути семантически подобны, если их термы (правые части подстановок) совпадают, что имеет место если ОО $[x_7]=[x_8]$ или критерий подобия $[x_7][x_8]^{-1}=1$ имеет место быть (см. $[y_8]$). Анализ ситуации (экспертом) подтверждает это и АКП(8-11) может быть редуцирован к одному из его путей, например, (8, 9, 11). В результате, получаем элементарный АКП(6, 7, 8, 9, 11) с путями подстановки (6, 7, 11) и (6, 8, 9, 11):

$$(6,7,11):\left\{\begin{array}{l}[y_5]=[x_4];\\ [y_6]=[x_4];\\ [y_5]=[x_5];\\ [y_6]=[x_6];\\ [y_8]=[x_7].\end{array}\right\}\quad (6,8,9,11):\left\{\begin{array}{l}[y_5]=[x_4];\\ [y_6]=[x_4];\\ [y_5]=[x_5];\\ [y_6]=[x_6];\\ [y_7]=[x_4]\ [x_4];\\ [y_7]=[x_5]\ [x_6];\\ [y_8]=[x_7].\end{array}\right\}$$

Убрав из этих путей совпадающие подстановки, получим:

$$(6,7,11):\left\{[y_7]=[x_3]\ [x_3]\right\}(6,8,9,11):\left\{\begin{array}{l}[y_7]=[x_4]\ [x_4];\\ [y_7]=[x_5]\ [x_6].\end{array}\right\}$$

Подстановку [y7] = [x3][x3] для пути (6, 7, 11) мы взяли из Y5 . Для выполнения редукции АКП(6, 7, 8, 9, 11) необходимо, чтобы выполнялись следующие равенства:

$[x_3]$ = $[x_4]$ – [объект из файла1] = [объект из файла2] ;
$[x_3]$ = $[x_5]$ – [объект из файла1] = [константа1];
$[x_3]$ = $[x_6]$ – [объект из файла1] = [константа2].

Если с равенством абстрактных сущностей, вводимых из файла и консоли, эксперт может согласиться, то их равенство с предопределенными константами, как альтернативы, требует углубленного анализа функциональности АКП (6-11). Анализировать данную ситуацию должен эксперт, а выявлять подобные ситуации - инструментальный комплекс (ИК).

Подобная процедура – рекурсивная. Первым редуцируется самый вложенный АКП, поэтому АКП и рассматриваются как элементы факторизации УГП. Выявленные в них отношения между X и Y передаются без искажений и потерь в соответствии с вложенностью и следованием АКП.

Если всем переменным из X соответствуют разные сущности, то с помощью теории подобия семантика функциональных связей определяется полностью и однозначно. Превентивное назначение абстрактных сущностей переменным из X и призвано обеспечить данный эффект. Более того, назначение однотипным, с точки зрения языка программирования, переменным из X разных абстракций позволяет их разграничивать как физические сущности и контролировать конкретные функциональные связи.

Например, в интересах поиска закладок определенные в программе константы должны интерпретироваться как *предопределенные значения* для *предопределённых действий*. Поэтому пути выполнения программы, в формировании которых участвуют константы, подлежат обязательной проверке. Семантику предопределённых действий как и альтернативных действий мы обязательно обнаружим, так как заданные $[x_k]$ при подстановках в процессе символьной интерпретации конкретизируют все связи.

Отличительной особенностью предлагаемого метода верификации и выявления закладок является вовлечение в процесс анализа семантических аспектов в контексте управляющих связей в программах, исключающих аппроксимацию или пропуск функциональных связей.

3. Динамическая верификация программ. Под динамической верификацией программ будем понимать контроль правильности семантики вычислений в ходе их реального выполнения. Понятно, что данные для динамического контроля необходимо предварительно получить, как получаем исполняемый код во время компиляции. Естественным образом напрашивается идея использовать для этого результаты статической верификации в виде паспорта π программы П. Поэтому

статическую верификацию должен проходит точно тот же исполняемый код, что и динамическую верификацию.

Паспорт π исполняемой программы П создается в процессе статической верификации (по дизассемблированному коду П) и содержит:

– вектор X с именами исходных данных;

– систему определяющих соотношений как систему линейных однородных уравнений $Az = 0$ с пространством столбцов, отображаемым на X:

– любое (желательно целочисленное) частное решение $Az = 0$ – вектор исходных абстрактных размерностей для X.

Для верификации исполняемой программы П в темпе реального времени выполнения нам понадобится интерпретирующий сопроцессор.

Множество команд *K* исполняемой программы П (команды ассемблера) разбивается на три подмножества:

K_A – аддитивные команды (сложение, вычитание, сравнение,…);

K_M – мультипликативные команды (умножение, деление, сдвиг,…):

K_N – не интерпретируемые команды .

Сопроцессор интерпретирует команды основного процессора следующим образом:

– если основной процессор выполняет команду $\kappa_i \in KA$, то сопроцессор выполняет сравнение размерностей ее операндов;

– если основной процессор выполняет операцию $\kappa_i \in KM$, то сопроцессор выполняет манипуляции с размерностями операндов (сложение или вычитание);

– если основной процессор выполняет операцию $\kappa_i \in KN$, то сопроцессор простаивает.

Данная интерпретация является прямой симуляцией действий, связанных с преобразованием степенных одночленов в линейные уравнения.

Выход исполняемой программы за пределы контролируемой среды (верифицированной в статике) определяется по результатам манипулирования операндами команд, для каждого из которых в памяти M^C сопроцессора отведено место и в него записано абстрактное паспортное (из зафиксированного в паспорте частного решения для $Az=0$) или сформированное в процессе интерпретации значение операнда.

Использование данных методов статической и динамической верификации перспективно для ПО замкнутых, не подверженных ди-

намическим изменениям программных систем. В отличие от [3] и [4] здесь связи по управлению контролируются опосредованно.

4. Заключение. Задача исчерпывающего выявления НДВ в программном коде, в общем случае, алгоритмически неразрешима, а существующие частные решения приходится аппроксимировать, чтобы избежать «комбинаторного взрыва» - сделать их сложность приемлемой для практической реализации. Чаще всего динамическую верификацию подменяют целенаправленным и многоаспектным тестированием. В предложенном методе динамическая верификация осуществляется как обязательная предопределенная процедура при каждом выполнении прошедшего статическую верификацию ПО. Данный подход позволяет исключить возможность выполнения кода не прошедшего статическую верификацию, в том числе внесенного позже в систему вредоносного кода.

Литература

1. *Седов Л.Н.* Методы подобия и размерности в механике // М: Наука. 1981. 448с.
2. *Ильин В. А., Позняк Э. Г.* Линейная алгебра // М.: ФИЗМАТЛИТ. 2004. 280 с.
3. *Компаниец Р.И., Ковалев В.В., Маньков Е.В.* Экспертиза и защита кода программ на основе автоматов динамического контроля // Защита информации. INSIDE. 2007. №3. С. 48–55.
4. Инструментального комплекс для автоматизации проведения статического и динамического анализа потоков управления в исполняемых кодах программ «IRIDA» // ТУ 425790.007.72410666.04. ООО «Газинформсервис».

References

1. Sedov L.N. *Metody podobiya i razmernosti v mehanike* [Methods of similarity and dimensionality in mechanics]. M: Nauka. 1981. 448 p. (In Russ.).
2. Ilin V.A., Poznyak E.G. *Lineinaya algebra* [Linear algebra]. M.: FIZMATLIT. 2004. 280 p. (In Russ.).
3. Kompaniez R.I., Kovalev V.V., Mankov E.V. [Expertise and protection of a program's code based on automata of dynamic control] *Zaschita informatzii.Inside – Information Security. Inside*. 2007. vol. 3. pp. 48–55 (In Russ.).
4. The tool kit for the automation of static and dynamic analysis of control flow in the executable program codes «IRIDA». TU 425790.007.72410666.04. OOO «Gazinformservice». (In Russ.).

Ковалев Виктор Васильевич — кандидат технических наук, профессор, доцент кафедры систем сбора и обработки информации Военно-космической академии имени А.Ф. Можайского. Область научных интересов: реверс инжиниринг, надежность, устойчивость функционирования и верификация программного обеспечения. Число научных публикаций — 14. ВКА имени А.Ф.Можайского, ул. Ждановская, д.13 г. Санкт-Петербург, 197198 , РФ, р.т. +7(812)347-9687.

Kovalyov Viktor Vasilevich — Ph.D., professor, associate Professor department systems collecting and processing information of Military space Academy named of A.F. Mozhayskiy. Research interests: reverse engineering; software reliability, stability and verification. The number of scientific publications — 14. MSA named of A.F. Mozhayskiy, st. Zhdanovskay, h.13 c. St. Petersburg, 197198, RF, office phone +7(812)347-9687.

Компаниец Радион Иванович — преподаватель кафедры систем сбора и обработки информации Военно-космической академии имени А.Ф. Можайского. Область научных интересов: построение трансляторов, верификация программного обеспечения. Число научных публикаций — 15. ВКА имени А.Ф. Можайского, ул. Ждановская, д.13 г. Санкт-Петербург, 197198 , РФ, р.т. +7(812)347-9687.

Kompaniets Radion Ivanovich — teacher of department systems collecting and processing information of Military space Academy named of A.F. Mozhayskiy. Research interests: construction of compilers; software verification. The number of scientific publications — 15. MSA named of A.F. Mozhayskiy, st. Zhdanovskay, h.13 c. St. Petersburg, 197198, RF, office phone +7(812)347-9687.

Новиков Владимир Александрович — к.т.н., докторант кафедры систем сбора и обработки информации Военно-космической академии имени А.Ф. Можайского. Область научных интересов: реверс инжиниринг, построение трансляторов, верификация программного обеспечения. Число научных публикаций — 34. ВКА имени А.Ф.Можайского, ул. Ждановская, д.13 г. Санкт-Петербург, 197198 , РФ, р.т. +7(812)347-9687.

Novikov Vladimir Aleksandrovich — Ph.D., doctoral candidate of the department systems collecting and processing information of Military space Academy named of A.F. Mozhayskiy. Research interests: reverse engineering; construction of compilers; software verification. The number of scientific publications — 34. MSA named of A.F. Mozhayskiy, st. Zhdanovskay, h.13 c. St. Petersburg, 197198, RF, office phone +7(812)347-9687.

РЕФЕРАТ

Ковалев В.В., Компаниец Р.И., Новиков В.А. **Верификация программ на основе соотношения подобия.**

В работе рассматриваются взаимосвязанные методы статической и динамической верификации программ, представленных в исполняемых кодах. Верификация осуществляется на основе проверки семантического подобия их поведения при достижении данного состояния по всем возможным путям, что исключает аппроксимацию функциональности верифицируемых программ. Снижение трудоемкости метода достигается отображением переменных программы на их абстрактные размерности.

В качестве моделей программ используется управляющий граф с факторизацией на артикулирующие компоненты и модели поведения альтернативных и повторяющихся фрагментов программы в процессе интерпретации программы в терминах абстрактных размерностей. Поведенческие модели представляются системами линейных однородных уравнений, анализ которых на совместность и непротиворечивость позволяет получить необходимые условия правильности функциональных связей в программах.

По результатам статической верификации строится паспорт программы, по которому с помощью сопроцессора можно осуществлять действительную динамическую верификацию данной программы в реальном времени при ее выполнении.

SUMMARY

Kovalev V.V., Kompanietc R. I., Novikov V.A. **Verification of Programs Based on Similarity Relations.**

This work represents interconnected methods of static and dynamic verification of the programs that are represented in source codes. Verification is based on the semantic similarity inspection of their behavior at the moment when they reach this state in all possible ways, which excludes functionality approximation of the verified programs. Labor intensiveness decrease is achieved by mapping program variables into their abstract dimensions.

Control graph with the factorization on the articulating components and behavioral models of alternative and repetitive fragments of the program in the interpretation of the program in terms of abstract dimensions are used as models of programs. Behavioral models are represented by systems of the linear homogeneous equations which compatibility and consistency analysis allows to obtain the necessary conditions of correctness of the functional connections in the programs.

Program passport is based on the results of the static verification, and which is used by the coprocessor to implement the valid dynamic verification of the running program.