УДК 004.056.53

О. Ф. Немолочнов, Л. Г. Осовецкий

КРИЗИС ПРОМЫШЛЕННОЙ ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ, НЕДЕКЛАРИРОВАННЫЕ ВОЗМОЖНОСТИ И *DON'T CARE*

Рассматриваются вопросы верификации вычислительных процессов по графоаналитическим моделям, управляемых частично-определенными булевыми функциями. Исследуются задачи поиска по булеву графу управления и кубическим покрытиям недекларированных возможностей и мертвого кода как следствия *don't care*. Приведены примеры построения покрытий для булева графа и верификация *don't care* в виде покрытия конъюнкции отношений-неравенств, тождественно равных нулю.

Ключевые слова: *технология программирования, недекларированные возможности, верификация, мертвый код, угрозы безопасности.*

Введение. Определим вычислительный процесс как процесс преобразования информации по формулам и алгоритмам, используемым для вычисления значений некоторого множества переменных. Вычисление значений переменных по разным формулам и алгоритмам осуществляется под управлением заданного множества неравенств-отношений, которые задают условия-предикаты, так как каждое отношение может либо выполняться, либо не выполняться, т.е. принимать два значения: *true* и *false*, и, следовательно, образует предикат на множестве значений переменных, входящих в левую и правую части неравенств. Для описания вычислительного процесса, порождаемого логической схемой или программой, необходимо построить модель, которая позволит формализовать его описание и упростить решение различных задач синтеза и анализа с применением аппарата теории множеств и алгебры логики. В качестве такой модели удобно использовать графоаналитическое представление вычислительного процесса [1]. В вершинах графа располагаются итеративные и рекуррентные формулы и условия-предикаты (отношения). Связи между вершинами задаются дугами. В общем случае в вершине может размещаться любой алгоритм преобразования информации. Дуги управления образуют конъюнкции условий-предикатов, их дизъюнкции образуют булевы функции. Основной задачей исследования вычислительного процесса является его верификация в соответствии с декларацией, состоящая в верификации декларированных и недекларированных возможностей процесса и в поиске несуществующих значений, образующих множество *don't care*.

Таким образом, задача верификации вычислительного процесса может быть сведена к поиску недекларированных возможностей (НДВ) и конъюнкций условий-предикатов, тождественно равных нулю, для которых системы неравенств не имеют решений. НДВ на графе вычислительного процесса образуют множество вершин и дуг, недостижимых через входные последовательности наборов данных, построенных по декларации и *don't care*, которые порождают частично-определенные булевы функции. Эти функции при их отображении на n-мерный двоичный куб E_n^2 могут быть заданы покрытиями комплексов $K^1(f)$, где f=1, $K^0(f)$, где f=0, и $K^d(f)$, где $f=d$ (*don't care*). Поиск и верификация вершин (конъюнкций) комплекса $K^0(f)$ и составляет основную задачу анализа вычислительных процессов.

Графоаналитическая модель вычислительного процесса. Информационные потоки в вычислительных процессах управляются некоторым множеством условий-предикатов в виде неравенств-отношений. Неравенства задают отношения между значениями переменных, входящих в их левую и правую части. Неравенства могут выполняться или не выполняться

и порождают, таким образом, предикат P, равный T при выполнении, и равный F — при невыполнении. Под формулой понимается конечная последовательность символов переменных и знаков математических операций между ними, т.е. рассматривается линейная формула FR. Значение, вычисленное по формуле FR, будем обозначать как |FR|.

Под алгоритмом понимается некоторая конечная последовательность элементарных действий, приводящих к однозначному результату — значению переменной. В общем случае обобщением формулы или алгоритма является некоторый оператор S, имеющий одну точку входа (T_{in}) и одну точку выхода (T_{out}).

Представление и описание вычислительного процесса в виде графа как множества вершин и связей между ними является некоторой метамоделью, не зависящей от конкретной реализации, например, в виде логической схемы или программы на алгоритмическом языке.

Для описания вычислительного процесса введем три типа вершин [2]: линейные — для формул и операторов; условные — для отношений-неравенств; объединения связей между ними. Связи между вершинами являются однонаправленными и, следовательно, образуют дуги, т.е. всегда имеется источник и приемник передачи управления, и, таким образом, информационные потоки в виде последовательности вершин и дуг являются направленными и детерминированными. Аналитическое описание вершин задается в виде логико-алгебраических выражений, сочетающих в себе логику условий-предикатов и алгебраические формулы вычисления переменных, а также может быть задано в виде кубических покрытий [3].

Далее вычислительный процесс может быть фрагментирован на замкнутые параллельные структуры (SR), реализующие по разным формулам альтернативные вычисления переменных r или различных переменных в зависимости от заданного множества условий-предикатов. Условия-предикаты могут задаваться либо впрямую булевыми переменными, либо косвенно через неравенства-отношения. Некоторое множество вершин и дуг графа вычислительного процесса образует параллельную структуру, если оно содержит одну точку входа T_{in} и одну точку выхода T_{out}. Для структуры SR все разветвления по условиям должны сходиться в одной точке T_{out}. На алгоритмическом языке высокого уровня им соответствуют условные операторы присвоения.

Любая булева функция f, входящая в параллельную структуру SR, является полностью заданной, т.е. образует комплекс $K(f)=K^1(f)\cup K^d(f)\cup K^0(f)$. Другими словами, если в SR содержится n условий, то любая булева функция f из SR может быть отображена на кубе E_n^2 множествами вершин K^1, где $f=1$, K^0, где $f=0$, и K^d, где $f=d$, таких что $K^1 \cup K^0 \cup K^d = E_n^2$. Здесь d (*don't care*) есть такие вершины куба E_n^2, где значение функции не определено вследствие того, что конъюнкции условий-предикатов дают пустые пересечения множеств значений переменных, входящих в неравенства-отношения, т.е. система неравенств не имеет решения.

Параллельные структуры SR могут находиться между собой в следующих отношениях:

— включение, когда некоторая структура SR_j содержит в себе некоторую структуру SR_i ($\text{SR}_j \supset \text{SR}_i$);

— конъюнкция — $\text{SR}_j \& \text{SR}_i$, т.е. последовательная композиция;

— дизъюнкция — $\text{SR}_j \vee \text{SR}_i$, т.е. параллельная композиция.

Так как любая структура SR является замкнутой, покрытия входящих в нее булевых функций можно (и нужно) строить независимо друг от друга, полагая, что условия-предикаты локализованы в ней, благодаря чему и достигается компактность покрытий. Таким образом, при построении покрытий для некоторой структуры SR другие структуры графа вычисли-

тельного процесса могут рассматриваться как некоторые нераскрываемые операторы S, что соответствует реализации вычислительного процесса, описанного на языке высокого уровня, в виде блоков и модулей.

На рис. 1 показан пример параллельной структуры, реализующей интервальную формулу (IFR), при $k_1 < k_2$:

$$r = \begin{cases} \text{IFR}_1 \text{ при } x < k_1; \\ \text{IFR}_2 \text{ при } k_1 \le x \le k_2, \\ \text{IFR}_3 \text{ при } x > k_2. \end{cases}$$

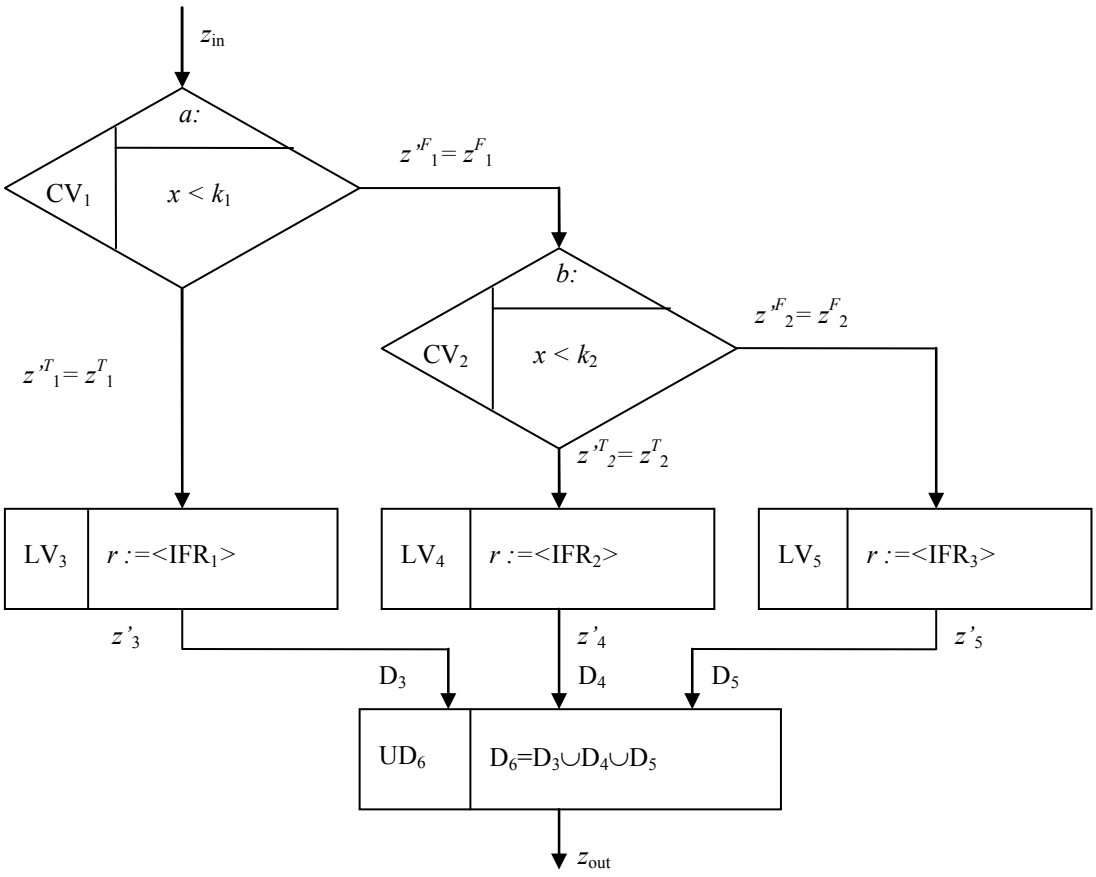

Рис. 1

На рисунке отношение $x<k_1$ обозначено через *a:* $x<k_1$, отношение $x>k_2$ — через *b:* $x>k_2$, и тогда соответственно $\bar{a}: x \ge k_1$ и $\bar{b}: x \le k_2$; CV — условная вершина, LV — линейная вершина, D — дуга, UD — объединение дуг, T и F — предикаты.

Выражение для переменной r на языке высокого уровня может быть записано условным оператором присваивания:

r:=if ($x<k_1$) *then* <IFR_1>
else if ($x>k_2$) *then* <IFR_3>
else <IFR_2>.

Результаты вычисления покрытий для переменной r сведены в табл. 1; в начале таблицы даны исходные покрытия для всех вершин графа в соответствии с типовыми покрытиями и в терминах обозначений, принятых в работе [2].

Таблица 1

z_{in}	a	b	r	r'	$z_1'^T$	$z_1'^F$	$z_2'^T$	$z_2'^F$	z_3'	z_4'	z_5'	z_{out}'	Примечание
									1	0	0	1	C_1
									0	1	0	1	C_2
									0	0	1	1	C_3
									0	0	0	0	C_4 } $C(\mathrm{UD}_6)$
			×	/IFR$_1$/									C_5
			×	×									C_6 } $C(\mathrm{LV}_3)$
			×	/IFR$_2$/				1		1			C_7
			×	×				0		0			C_8 } $C(\mathrm{LV}_4)$
			×	/IFR$_3$/			1				1		C_9
			×	×			0				0		C_{10} } $C(\mathrm{LV}_5)$
		0				1	0	1					C_{11}
		1				1	1	0					C_{12}
		×				0	0	0					C_{13} } $C(\mathrm{CV}_2)$
1	0				0	1							C_{14}
1	1				1	0							C_{15}
0	×				0	0							C_{16} } $C(\mathrm{CV}_1)$

В табл. 2 приведены пересечения кубов, имеющих непустое значение. Вычисления проводились от z'_{out} к z_{in} и, фактически, свелись к перебору четырех кубов из покрытия $C(\mathrm{UD}_6)$.

Таблица 2

z_{in}	a	b	r	r'	$z_1'^T$	$z_1'^F$	$z_2'^T$	$z_2'^F$	z_3'	z_4'	z_5'	z_{out}'	Примечание
1	1	×	×	/IFR$_1$/	1	0	0	0	1	0	0	1	$\boldsymbol{C_1 \cap C_5 \cap C_{15} \cap C_8 \cap C_{10}}$
1	0	0	×	/IFR$_2$/	0	1	0	1	0	1	0	1	$\boldsymbol{C_2 \cap C_7 \cap C_{11} \cap C_6 \cap C_{10}}$
1	0	1	×	/IFR$_3$/	0	1	1	0	0	0	1	1	$\boldsymbol{C_3 \cap C_9 \cap C_{12} \cap C_6 \cap C_8}$
0	×	×	×	×	0	0	0	0	0	0	0	0	$\boldsymbol{C_4 \cap C_6 \cap C_8 \cap C_{10}}$

Удалив из покрытия промежуточные значения z', получим покрытие $C(r)$ в формате: $z_{in}\, a\, b\, r\, r'\, z'_{out}$:

$$C(r)=\left\{\begin{array}{cccc|cc} z_{in} & a & b & r & r' & z'_{out} \\ 1 & 1 & \times & \times & |\,\mathrm{IFR}_1| & 1 \\ 1 & 0 & 0 & \times & |\,\mathrm{IFR}_2| & 1 \\ 1 & 0 & 1 & \times & |\,\mathrm{IFR}_3| & 1 \\ 0 & \times & \times & \times & \times & 0 \end{array}\right\},$$

структура которого соответствует структуре покрытий для типовых вершин и описывает режим вычисления по различным итеративным формулам и режим хранения $r=\times$.

Поиск и верификация недекларированных возможностей. *Недекларированные возможности* — функциональные возможности программного обеспечения (ПО), не описанные или не соответствующие описанным в документации, при использовании которых возможно нарушение конфиденциальности, доступности или целостности обрабатываемой информации. Реализацией недекларированных возможностей, в частности, являются программные закладки.

Программные закладки — преднамеренно внесенные в ПО функциональные объекты, которые при определенных условиях (входных данных) инициируют выполнение не описанных в документации функций ПО, приводящих к нарушению конфиденциальности, доступности или целостности обрабатываемой информации [4].

Определим недекларированные возможности в общем виде как все особенности и возможности вычислительного процесса (ВП), порождаемого программой в ходе ее исполнения, не описанные в декларации. Природа НДВ носит как объективный, так и субъективный ха-

рактер в силу сложности и размерности самого программного продукта (ПП). Однако можно выделить, в общих чертах, основные источники и составляющие части НДВ:

1) наличие в ПП специально предусмотренных закладок для наблюдения за ВП и его отладки в ходе проектирования и реализации;

2) исключение излишне мелких подробностей ВП при составлении декларации из-за стремления к ее компактности;

3) наличие в частично-определенных булевых функциях f управления ВП неопределенных значений d в виде комплекса $K^d(f)$, который объективно существует, но, как правило, не используется и не описывается.

Поиск и верификацию НДВ можно осуществить путем моделирования вычислительного процесса либо в виде программы, либо на более высоком уровне — в виде графоаналитической модели.

Моделирование ВП проводится с использованием испытательных наборов данных из его декларации посредством прямого исполнения программы в ходе тестовых экспериментов. В ходе экспериментов необходимо фиксировать вершины и дуги, которые достижимы на входе T_{in}, вычисляются и наблюдаются через выход T_{out} в исследуемой программе.

О п р е д е л е н и е . Множество вершин и дуг графоаналитической модели вычислительного процесса образуют множество НДВ, если они недостижимы в результате тестовых экспериментов, построенных по декларации.

Множество задействованных вершин и дуг графа образуют декларированные возможности (ДВ) ВП и соответственно программы его реализующей. Таким образом, все множество вершин и дуг $M(\text{V,D})$ распадается на два подмножества $M^{\text{ДВ}}$ и $M^{\text{НДВ}}$: $M(\text{V,D})=M^{\text{ДВ}}\cup M^{\text{НДВ}}$. Указанные подмножества можно построить путем моделирования константных неточностей условий-предикатов ($m^1\equiv 1$ (T) и $m^0\equiv 0$ (F)), непосредственно вводя их либо в программу [5], либо в условные вершины графоаналитической модели ВП, при условии, что такая модель построена. С этой целью в ходе тестовых экспериментов производится сравнение результатов вычислений при отсутствии неточностей и с заданной неточностью $m^p\in M$, где M — все множество условий-предикатов. Условие-предикат может полностью не проверяться или проверяться только в одном из значений T или F. В этом смысле множество непроверяемых неточностей и формирует множество вершин и дуг, образующих НДВ.

Поиск и верификация *don't care.* Для любой булевой функции f от n переменных область определения состоит из 2^n их значений, задающих канонические формы f. Это множество значений может быть сопоставлено с вершинами n-мерного двоичного куба E_n^2; для упрощения логических выражений можно применять исчисление кубических комплексов путем построения неизбыточных покрытий в виде дизъюнктивных нормальных форм (простых импликант) или в виде скобочных форм (полученных методом функциональной декомпозиции). Первый способ применяется при реализации ВП в виде логических схем, а второй — в виде ПП.

В случае частично-определенных булевых функций область определения из 2^n значений разбивается на два подмножества M^1 и M^d: $M^1\cup M^d=E_n^2$ и $M^1\cap M^d=\varnothing$. Множество M^1 состоит из конъюнкций, которые могут быть вычислены в значениях, равных единице, а множество M^d состоит из конъюнкций, тождественно равных нулю. Значения функции f на конъюнкциях из M^d принято обозначать как d, образующие значение f в виде *don't care*, т.е. значение функции f „безразличное“ и может быть произвольно присоединено к любой функции управления z, построенной на множестве M^1.

В случае если булевы переменные задаются в виде отношений-неравенств, то значениям d соответствуют системы неравенств, для которых отсутствуют решения, и соответственно множество отношений между переменными, входящими в левую и правую части неравенств, дают пустое пересечение.

Итак, определим множество конъюнкций *don't care* как множество конъюнкций, тождественно равных нулю. Поиск таких конъюнкций может быть осуществлен либо непосредственным решением систем неравенств в аналитической форме, либо методом моделирования отношений на числовой оси в виде линий Ламберта или на плоскости в виде диаграмм Эйлера — Венна непосредственным заданием этих множеств [6].

Верификация *don't care* состоит в построении покрытий C^1 для M^1 и C^d для M^d, которые позволяют производить поиск *don't care* в сокращенной форме для подмножеств из M^d и, следовательно, сократить перебор при решении систем неравенств-отношений.

Рассмотрим описанные выше положения на примере. Пусть на числовых осях x и y заданы четыре отношения:

$$a: x < 4, \quad b: x > 5, \quad c: y < 3, \quad d: y > 6.$$

Здесь отношения a и b не зависят от значений c и d, поэтому поиск *don't care* можно произвести раздельно.

Итак, если $x<4$ и $x>5$, то эта система неравенств не имеет решения, т.е. на числовой оси x невозможно найти значения x, удовлетворяющие конъюнкции $a=b=1$, т.е. $ab \equiv 0$, которое и есть *don't care*. Аналогично, конъюнкция $c=d=1$ также не имеет решения и $cd \equiv 0$ образует *don't care* для $y<3$ и $y>6$.

Покрытие множества M^d относительно множества отношений $\{a,b,c,d\}$ компактно можно записать в виде двух кубов:

$$C^d = \left\{ \begin{array}{cccc} a & b & c & d \\ \hline 1 & 1 & \times & \times \\ \times & \times & 1 & 1 \end{array} \right\}.$$

Аналогично можно для верификации *don't care* построить покрытие для M^1 в виде трех кубов:

$$C^1 = \left\{ \begin{array}{cccc} a & b & c & d \\ \hline \times & 0 & \times & 0 \\ 0 & \times & 0 & \times \\ \times & 0 & 0 & \times \end{array} \right\}.$$

Заметим, что $C^1 \cup C^d = E_n^2$ и $C^1 \cap C^d = \varnothing$, что и свидетельствует о рассмотрении всех конъюнкций для отношений $\{a,b,c,d\}$.

Указанные решения проиллюстрированы на рис. 2.

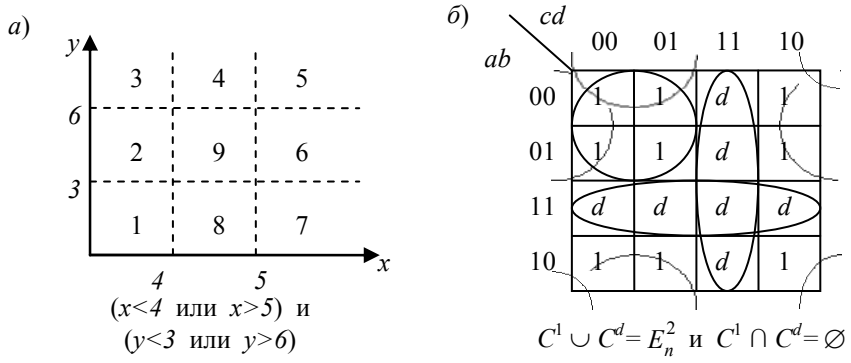

Рис. 2

Так, на рис. 2, *а* показаны отношения $\{a,b,c,d\}$ в системе координат xy: видно, что число существующих решений для системы неравенств равно 9, а оставшиеся 7 отношений решений не имеют. На развертке куба E_n^2 в виде карты Карно (рис. 2, *б*) вершины множества M^1 обозначены как 1, а множества M^d — как d.

Из рассмотренного примера видно, что задача поиска и верификации *don't care* состоит в переборе всех вариантов решений. Этот перебор может быть сокращен, если решения искать сразу в кубической форме.

Мертвый код как следствие *don't care*. Определим понятие мертвого кода (МК) как множество команд программы, которые не могут быть исполнены ни при каких значениях конъюнкций условий-предикатов. Это может быть только в том случае, если данные конъюнкции тождественно равны нулю. Такие конъюнкции и образуют множества *don't care* и порождают частично-определенные булевы функции. Если они запрограммированы в явном виде, то операторы программы, заданные только через них, никогда не будут исполняться. Иными словами, мертвый код всегда есть следствие *don't care*.

Такие конъюнкции могут быть заданы на уровне булевых переменных в виде явных тавтологий, например, $a \vee \overline{a}$ и $\overline{a\overline{a}}$, или в виде избыточных выражений, например, $a \vee ab$, $a \vee \overline{a}b$ и т.п. Указанная избыточность может быть следствием ошибок при программировании условий, например при настройке стандартных шаблонов, и если это ошибка, а не умысел, то она должна быть устранена путем минимизации логических выражений. Заметим, что для настраиваемых шаблонов *don't care* может быть только вычислено и указано, в противном случае теряется сам смысл шаблонов как стандартного и апробированного решения.

Если же *don't care* запрограммировано сознательно в виде артефакта, то порождаемые операторы образуют закладки. Эти закладки не могут быть обнаружены через тестовые эксперименты по полной декларации, т.е. декларации, содержащей декларированные и недекларированные возможности программы. В этом и состоит принципиальное различие между мертвым кодом и НДВ.

Мертвый код может быть в неявном виде задан при программировании отношений-неравенств, в этом случае необходимо осуществлять поиск *don't care* через решение систем неравенств. Для простоты рассмотрения приведем примеры мертвого кода в виде условных выражений над булевыми переменными. Пусть заданы условные выражения, реализующие операторы S_1 и S_2, которые могут быть операторами присваивания, перехода, обращения к процедурам или, в общем случае, любыми, в свою очередь, составными операторами, модулями или блоками. Итак, пусть заданы два условных выражения:

$$if\ (a\overline{a})\ then\ S_1\ else\ S_2 \quad \text{и} \quad if\ (a \vee \overline{a})\ then\ S_1\ else\ S_2.$$

В первом выражении никогда не будет исполняться оператор S_1, а во втором — оператор S_2. Операторы S_1 и S_2 в данном контексте могут рассматриваться как специально сделанные закладки. Конъюнкции, тождественно равные нулю, показаны на рис. 3 в виде функций управления z_i на булевых графах, реализующих приведенное выше условное выражение.

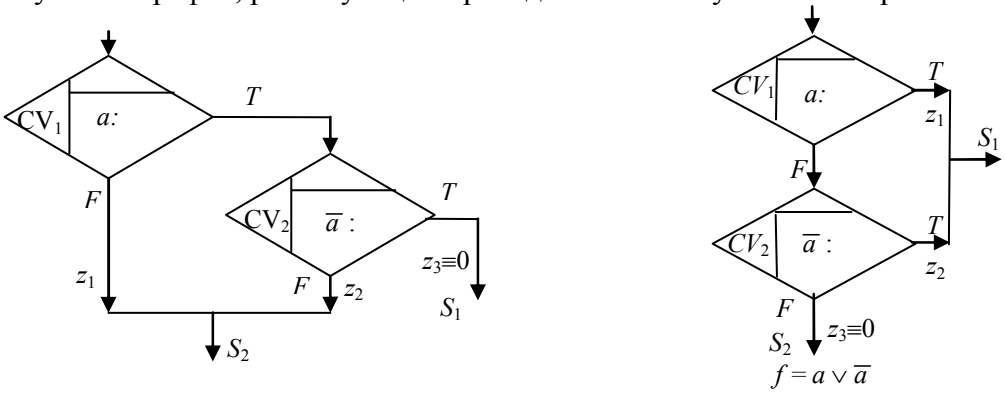

Рис. 3

Заключение. Наличие в вычислительном процессе, порождаемом программой при интерпретации ее команд процессором, недекларированных возможностей или мертвого кода несет в себе явную (НДВ) или неявную (МК) угрозу безопасности программному продукту.

Эта угроза может быть реализована, например, в виде компьютерного вируса. Поэтому поиск и верификация НДВ и МК является основной задачей в области защиты информации. Решение этой задачи конструктивно может быть найдено путем построения графоаналитической модели или, в частном случае, построения булева графа для функции управления вычислительным процессом. Для сокращения размерности задачи поиска НДВ и МК вычислительный процесс на графоаналитической модели следует разбивать на параллельные структуры SR. Такое разбиение позволяет локализовать поиск и верификацию НДВ и МК в рамках отдельно взятой параллельной структуры и не рассматривать все множество реализованных в программе условий-предикатов совместно.

СПИСОК ЛИТЕРАТУРЫ

1. *Немолочнов О. Ф., Зыков А. Г., Поляков В. И.* Комплексные кубические покрытия и графоаналитические модели как средство описания вычислительных процессов программ // Тр. Междунар. науч.-техн. конф. „Интеллектуальные системы“ (AIS’06) и „Интеллектуальные САПР“ (CAD-2006). М.: Физматлит, 2006. Т 2.

2. Модель и примитивы покрытий вершин циклических вычислительных процессов / *О. Ф. Немолочнов, А. Г. Зыков, Л. Г.Осовецкий, В. И. Поляков* // Изв. вузов. Приборостроение. 2007. Т. 50, № 8. С. 18—23.

3. *Немолочнов О. Ф., Зыков А. Г., Поляков В. И.* Кубические покрытия логических условий вычислительных процессов программ // Научно-технический вестник СПбГУ ИТМО. 2004. № 14. С. 225—233.

4. Руководящий документ Гостехкомиссии России „Защита от несанкционированного доступа к информации. Часть 1. Программное обеспечение средств защиты информации. Классификация по уровню контроля отсутствия недекларированных возможностей“. Введ. 04.06.1999 г.

5. Тестирование логических неисправностей вычислительных процессов в программах / *О. Ф. Немолочнов, А. Г. Зыков, Л. Г.Осовецкий, В. И. Поляков, К. В. Петров* // Информационные технологии. 2007. № 12. С. 2—5.

6. *Кондаков Н. И.* Логический словарь. М.: Наука. 1971. 656 с.

Сведения об авторах

Олег Фомич Немолочнов — д-р техн. наук, профессор; Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики, кафедра информатики и прикладной математики; E-mail: nof@ipmi.ifmo.ru

Леонид Георгиевич Осовецкий — д-р техн. наук, профессор; Центральный научно-исследовательский институт связи, Санкт-Петербург; E-mail: leoned.osovetsky@gmail.com

Рекомендована кафедрой информатики и прикладной математики НИУ ИТМО

Поступила в редакцию 01.07.13 г.