УДК 004.056.53

О. Ф. НЕМОЛОЧНОВ, А. Г. ЗЫКОВ, В. С. КУЛАГИН, Л. Г. ОСОВЕЦКИЙ,
В. И. ПОЛЯКОВ, А. В. СУХАНОВ

МЕТОД ОБНАРУЖЕНИЯ НЕДЕКЛАРИРОВАННЫХ ВОЗМОЖНОСТЕЙ И ЗНАЧЕНИЙ *DON'T CARE* ВЫЧИСЛИТЕЛЬНОГО ПРОЦЕССА

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 значения *don't care*. Приведены примеры построения покрытий для булева графа и верификации значений *don't care* в виде покрытия конъюнкции отношений-неравенств, тождественно равных нулю.

Ключевые слова: *вычислительный процесс, недекларированные возможности, мертвый код, графоаналитическая модель.*

Введение. Определим вычислительный процесс как процесс преобразования информации по формулам и алгоритмам, позволяющим вычислять значения некоторого множества переменных. Вычисление значений переменных по разным формулам и алгоритмам осуществляется в соответствии с множеством отношений-неравенств, которыми задаются условия-предикаты. Каждое отношение может либо выполняться, либо не выполняться, т.е. принимать два значения: *true* и *false* (верное и ложное), 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. Основная задача исследования вычислительного процесса (ВП) — его верификация в соответствии с декларацией — заключается в верификации декларированных и недекларированных возможностей ВП и в поиске несуществующих значений *don't care* (от англ. — букв.: „все равно“, „не заботит“).

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 последовательности входных наборов, построенных в соответствии с декларацией и значением *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 комплекса $K(f)$ и составляет основную задачу анализа вычислительных процессов.

Графоаналитическая модель вычислительного процесса. Информационные потоки в вычислительных процессах управляются некоторым множеством условий-предикатов в виде отношений-неравенств. Неравенства могут выполняться, порождая предикат P, равный T, или

не выполняться, порождая предикат F. Формула понимается как конечная последовательность переменных и знаков математических операций между ними, т.е. как линейная формула FR. Значение, вычисленное по формуле FR, будем обозначать как |FR|. Алгоритм понимается как некоторая конечная последовательность элементарных действий, приводящих к однозначному результату — значению переменной. 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 — это некоторая не зависящая от конкретной реализации метамодель, например, в виде логической схемы или программы на алгоритмическом языке.

Для описания вычислительного процесса введем три типа вершин [2]: 1) линейные — для формул и операторов; 2) условные — для отношений-неравенств; 3) объединения связей между линейными и условными вершинами. Связи между вершинами являются однонаправленными, т.е. всегда имеется источник и приемник передачи управления: таким образом, информационные потоки в виде последовательности вершин и дуг являются направленными и детерминированными. Аналитическое описание вершин может быть задано как в виде логико-алгебраических выражений, сочетающих в себе логику условий-предикатов и алгебраические формулы вычисления переменных, так и в виде кубических покрытий [3].

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

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 структуре SR содержится n условий, то любая булева функция f из SR может быть отображена на кубе E_n^2 множествами вершин K, таких что $K^1 \cup K^0 \cup K^d = E_n^2$. Здесь *d (don't care)* — такие вершины куба E_n^2, в которых значение функции не определено вследствие того, что конъюнкции условий-предикатов дают пустые пересечения множеств значений переменных, входящих в отношения-неравенства, 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.

Вследствие того, что 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, чем и достигается компактность покрытий. Таким образом, при построении покрытий для некоторой структуры SR другие структуры графа вычислительного 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 итеративную формулу (при $k_1 < k_2$):

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

На рис. 1 обозначение a соответствует отношению $x<k_1$, обозначение b — отношению $x>k_2$, тогда $\overline{a} : x \ge k_1$ и $\overline{b} : x \le k_2$. Остальные использованные на рис. 1 обозначения соответствуют принятым в работе [2].

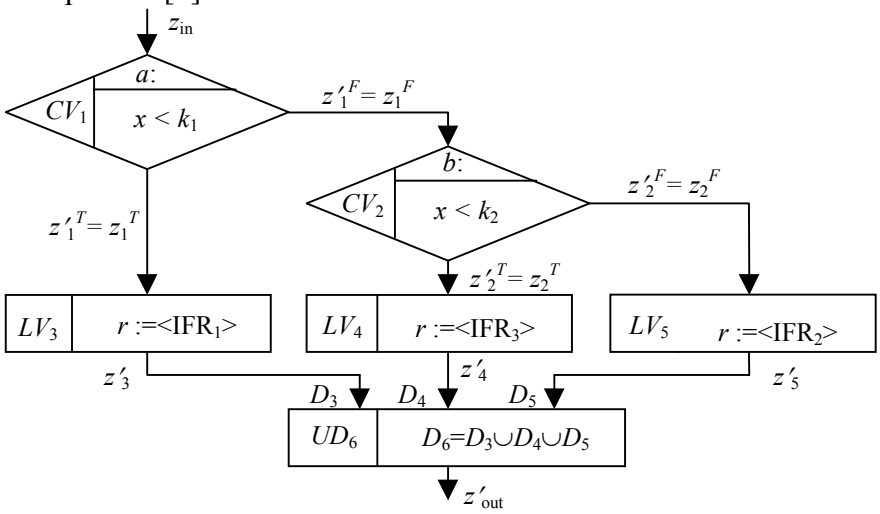

Рис. 1

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

r:= если ($x<k_1$), то <IFR$_1$>,
иначе если ($x>k_2$), то <IFR$_3$>,
иначе <IFR$_2$>.

Вычисления покрытия для переменной r сведены в табл. 1, где приведены исходные покрытия для всех вершин графа в соответствии с типовыми покрытиями и принятыми в работе [2] обозначениями.

Таблица 1

z_{in}	a	b	r	r'	z'^T_1	z'^F_1	z'^T_2	z'^F_2	z'_3	z'_4	z'_5	z'_{out}	Примечание
									1	0	0	1	C_1
									0	1	0	1	C_2 } $C(UD_6)$
									0	0	1	1	C_3
									0	0	0	0	C_4
			×	<IFR$_1$>	1				1				C_5
			×	×	0				0				C_6 } $C(LV_3)$
			×	<IFR$_2$>		1		1			1		C_7
			×	×		0		0			0		C_8 } $C(LV_5)$
			×	<IFR$_3$>		1	1			1			C_9
			×	×		0	0			0			C_{10} } $C(LV_4)$
		0				1	0	1					C_{11}
		1				1	1	0					C_{12} } $C(CV_2)$
		×				0	0	0					C_{13}
1	0				0	1							C_{14}
1	1				1	0							C_{15} } $C(CV_1)$
0	×				0	0							C_{16}

В табл. 2 приведены пересечения кубов, имеющих непустое значение. Вычисления производились от выхода z'_{out} ко входу z_{in} и свелись фактически к перебору четырех кубов из покрытия $C(UD_6)$.

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	$C_1 \cap C_5 \cap C_{15} \cap C_8 \cap C_{10}$
1	0	0	×	<IFR$_2$>	0	1	0	1	0	1	0	1	$C_3 \cap C_7 \cap C_{11} \cap C_6 \cap C_{10}$
1	0	1	×	<IFR$_3$>	0	1	1	0	0	0	1	1	$C_2 \cap C_9 \cap C_{12} \cap C_6 \cap C_8$
0	×	×	×	×	0	0	0	0	0	0	0	0	$C_4 \cap C_6 \cap C_8 \cap C_{10}$

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

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

Структура этого покрытия соответствует структуре покрытий для типовых вершин и описывает режим вычисления переменной r 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. Реализацией недекларированных возможностей являются, в частности, 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 не описанные в декларации особенности и возможности вычислительного процесса, порождаемого программой в ходе ее исполнения. Природа НДВ носит как объективный, так и субъективный характер в силу сложности и размерности самого программного продукта. Однако для НДВ можно выделить в общих чертах следующие основные источники и составляющие:

— наличие в программном продукте специально предусмотренных закладок для наблюдения и отладки ВП в ходе его проектирования и эксплуатации;

— „замалчивание" излишне мелких подробностей ВП при составлении декларации вследствие стремления к ее компактности;

— 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 ВП можно осуществить путем моделирования процесса либо в виде исполнения программы, либо — на более высоком уровне — по графоаналитической модели.

Моделирование ВП проводится на испытательных наборах входных переменных, построенных по его декларации путем прямого исполнения программы в ходе тестовых экспериментов. При этом необходимо фиксировать вершины и дуги графа, которые достижимы из точки T_{in}, вычисляются и наблюдаются в точке T_{out} исследуемой программы.

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

Множество задействованных вершин и дуг графа образуют декларированные возможности (ДВ) ВП и соответственно программы, его реализующей. Таким образом, все множество вершин и дуг $M(V, D)$ распадается на два подмножества $M^{ДВ}$ и $M^{НДВ}$: $M(V, D)=M^{ДВ}\cup M^{НДВ}$. Указанные подмножества можно построить путем моделирования константных „неисправностей“ для условий-предикатов ($m^1\equiv 1(T)$ и $m^0\equiv 0(F)$) непосредственным внесением 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 без „неисправностей“ и с заданной „неисправностью“ $m^p\in N$, где N — все множество условий-предикатов. Условие-предикат может быть полностью не проверено или проверено только в одном из значений T или F. В этом смысле множество непроверяемых „неисправностей“ и образует множество вершин и дуг НДВ.

Поиск и верификация значений *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 программного продукта.

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 аргументов, на которых функция принимает значение, равное единице, а множество M^d состоит из конъюнкций, тождественно равных нулю. Значения функции f на конъюнкциях из множества M^d принято обозначать как d — *don't care*: т.е. все равно какое значение имеет функция f, оно 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, следовательно, 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 путем непосредственного решения систем неравенств в аналитической форме, либо методом моделирования отношений на числовой оси в виде линий Ламберта или непосредственным заданием множеств на плоскости в виде диаграмм Эйлера — Венна [6].

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

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

$$\alpha: x<4,\quad \beta: x>5,\quad \gamma: y<3,\quad \zeta: y>6\,.$$

Здесь отношения α и β не зависят от значений γ и ζ, поэтому поиск значения *don't care* можно произвести раздельно.

Итак, если $x<4$ и $x>5$, то эта система неравенств не имеет решения: на числовой оси x невозможно найти значения x, удовлетворяющие конъюнкции $\alpha=\beta=1$, т.е. $\alpha\beta \equiv 0$, что и является значением *don't care*. Аналогично конъюнкция $\gamma=\zeta=1$ также не имеет решения, и тождество $\gamma\zeta \equiv 0$ образует значение *don't care* для $y<3$ и $y>6$.

Покрытие множества M^d относительно множества отношений $\{\alpha,\beta,\gamma,\zeta\}$ компактно можно записать в виде двух кубов:

$$C^d = \left\{ \begin{array}{cccc} \alpha & \beta & \gamma & \zeta \\ \hline 1 & 1 & \times & \times \\ \times & \times & 1 & 1 \end{array} \right\}.$$

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

$$C^1 = \left\{ \begin{array}{cccc} \alpha & \beta & \gamma & \zeta \\ \hline \times & 0 & \times & 0 \\ 0 & \times & 0 & \times \\ \times & 0 & 0 & \times \\ 0 & \times & \times & 0 \end{array} \right\}.$$

Заметим, что $C^1 \cup C^d = E_n^2$ и $C^1 \cap C^d = \varnothing$: это и свидетельствует о рассмотрении всех конъюнкций для отношений $\{\alpha,\beta,\gamma,\zeta\}$.

Рассмотренные решения проиллюстрированы на рис. 2. На рис. 2, *а* показаны отношения $\{\alpha,\beta,\gamma,\zeta\}$ в системе координат $x\,y$: видно, что число существующих решений для системы неравенств равно 9, а 7 отношений решений не имеют. На рис. 2, *б* показана развертка куба E_n^2 в виде карты Карно: 1 — вершины множества M^1, d — множества M^d.

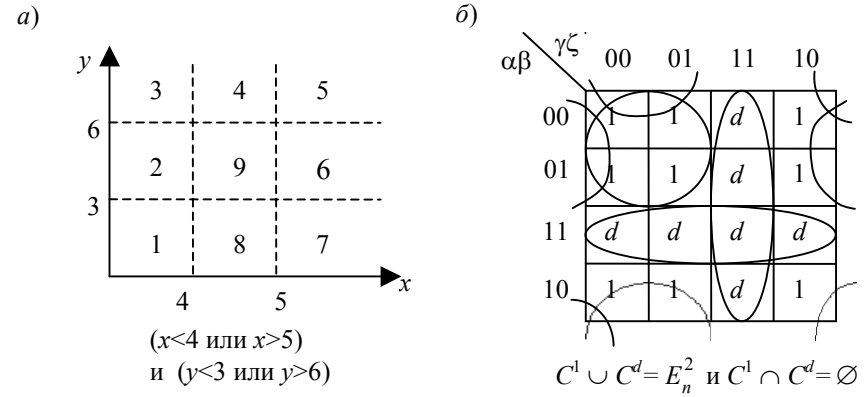

Рис. 2

Как следует из рассмотренного примера, задача поиска и верификации значения *don't care* заключается в переборе всех вариантов решений, этот перебор может быть сокращен, если решения искать сразу в кубической форме.

Мертвый код как следствие значения *don't care*. Определим понятие мертвого кода (МК) как множества 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 значений *don't care* и порождают частично-определенные булевы функции. Если функции запрограммированы в явном виде, то операторы, заданные в программе только через них, никогда не будут исполняться. Иными словами, мертвый код всегда есть следствие значения *don't care*.

Такие конъюнкции могут быть заданы на уровне булевых переменных в виде явных тавтологий, например, $a \vee \overline{a}$, $\overline{a\overline{a}}$ 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 значение *don't care* может быть только вычислено и указано, в противном случае теряется сам смысл шаблонов как стандартного и апробированного решения.

Если же значение *don't care* запрограммировано сознательно в виде артефакта, то порождаемые операторы образуют закладки. Эти закладки не могут быть обнаружены в ходе тестовых экспериментов по полной 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 значения *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 либо в общем случае любыми составными операторами, модулями или блоками. Итак, пусть заданы два условных выражения:

$$\text{если } (a\overline{a}), \text{ то } S_1, \text{ иначе } S_2,$$
$$\text{если } (a \vee \overline{a}), \text{ то } S_1, \text{ иначе } S_2.$$

В первом выражении оператор S_1 никогда не будет исполняться, а во втором не будет исполняться 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, *а*, *б* в виде функций управления z_i на булевых графах, реализующих приведенные выше условные выражения соответственно.

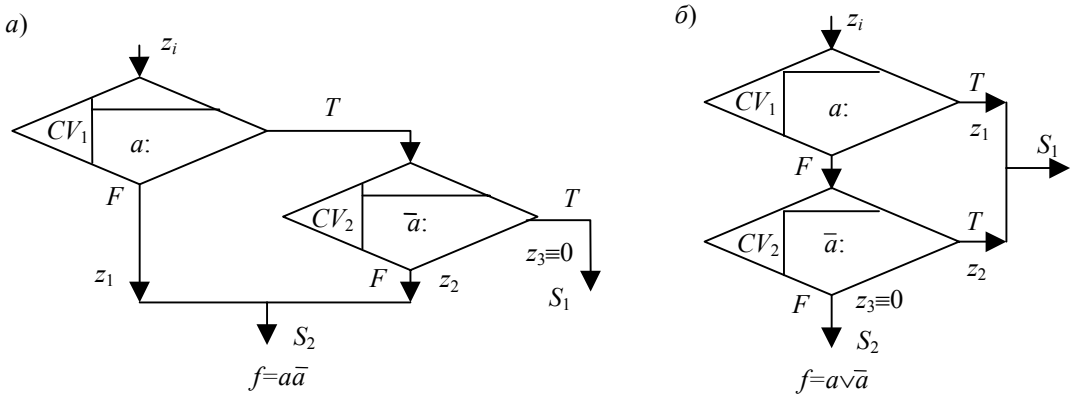

Рис. 3

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, построением 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. 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. С. 3—7.

2. Модель и примитивы 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 и программ // Науч.-техн. вестн. СПбГУ ИТМО. Информационные технологии, вычислительные и управляющие системы. СПб.: СПбГУ ИТМО, 2004. Вып. 14. С. 225—233.

4. Руководящий документ Гостехкомиссии России „Защита от несанкционированного доступа к информации. Ч. 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* и др. // Информ. технологии. 2007. № 12. С. 2—5.

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

Сведения об авторах

Олег Фомич Немолочнов — д-р техн. наук, профессор; Санкт-Петербургский государственный университет информационных технологий, механики и оптики, кафедра информатики и прикладной математики

Анатолий Геннадьевич Зыков — канд. техн. наук, доцент; Санкт-Петербургский государственный университет информационных технологий, механики и оптики, кафедра информатики и прикладной математики;
E-mail: zykov_a_g@mail.ru

Вячеслав Сергеевич Кулагин — канд. техн. наук, доцент; Санкт-Петербургский государственный университет информационных технологий, механики и оптики, кафедра электроники

Леонид Георгиевич Осовецкий — д-р техн. наук, профессор; Санкт-Петербургский государственный университет информационных технологий, механики и оптики, кафедра безопасных информационных технологий

Владимир Иванович Поляков — канд. техн. наук, доцент; Санкт-Петербургский государственный университет информационных технологий, механики и оптики, кафедра вычислительной техники;
E-mail: v_i_polyakov@mail.ru

Андрей Вячеславович Суханов — канд. техн. наук, доцент; Санкт-Петербургский государственный университет информационных технологий, механики и оптики, кафедра безопасных информационных технологий

Рекомендована кафедрой
информатики и прикладной математики

Поступила в редакцию
06.06.09 г.