АЛГОРИТМ ДИАГНОСТИРОВАНИЯ SOFTWARE МОДУЛЯ

Хаханов В.И., Сушанов А.В., Давыдов М.Д., Литвинова Е.И.
Харьковский национальный университет радиоэлектроники
Украина, 61166, Харьков, пр. Ленина 14
Тел., факс: (057) 702-13-26, E-mail: hahanov@kture.kharkov.ua

Software product testing methods based on disjunctive normal forms are offered. Normal forms represent fault coverage table.

1. Описание алгоритма диагностирования

При разработке программных продуктов большого объёма актуальной задачей является проверка создаваемого проекта на корректность выполняемых операций. Сложное программное средство содержит огромное количество ветвлений, и проверить безошибочное выполнение программы на каждой из логических путей представляет собой достаточно сложную задачу. Далее на конкретном примере рассматривается метод поиска некорректных операторов (ошибок или дефектов) в программном средстве, основанный на представлении алгоритма программы в виде графовой структуры в целях последующего создания тестов и диагностирования дефектов. Пусть необходимо верифицировать программное средство, позволяющее вычислять следующую сумму функций:

$$S = (x) + \omega(x),$$

$$x = \begin{cases} x + 3; & x < 2; \\ 2x - 3; & 2 \le x < 12; \\ -3x + 7; & x \ge 12; \end{cases} \quad \omega(x) = \begin{cases} \sin(x + \pi/3), x < 2\pi/3; \\ \sin(\pi x) + 2, x \ge 2\pi/3. \end{cases}$$

Один из возможных вариантов решения задачи на языке C++ представлен следующим листингом:

```
#include <iostream>   #include <math.h>
using namespace std;
int main()
{    const double Pi=3.14159;
     double F, w, f, x;
     cin>>x;
if (x<2) f=x+3;
else if ((x>=2) && (x<12)) f=2*x-3;
else f=-3*x+7;
if (x<2./3.*Pi)
w=sin(x+Pi/3);
else w=sin(Pi*x)+2;
F=f+w;
cout<<F<<endl;
     return 0; }
```

Пусть ошибка содержится в одном из операторов вычислительной части программы. else w=sin(Pi*x) - 2;

Необходимо определить некорректный оператор в программном коде на основе технологии тестирования, использующей граф-модель кода. Стадии диагностирования программного кода включают 4 процедуры, представленные на рис. 1.

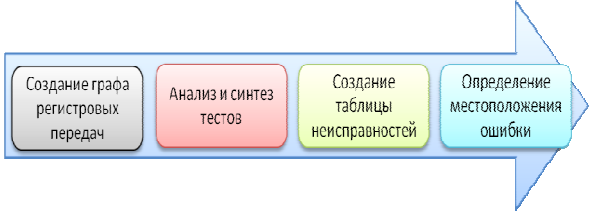

Рис. 1. Технология тестирования программного кода

2. Построение графа регистровых передач

Дуги в графе представляют собой совокупность фрагментов кода или отдельных операций (рис. 2), а вершинами являются точки мониторинга информации (регистры, переменные, память), которые также используются для формирования ассерций.

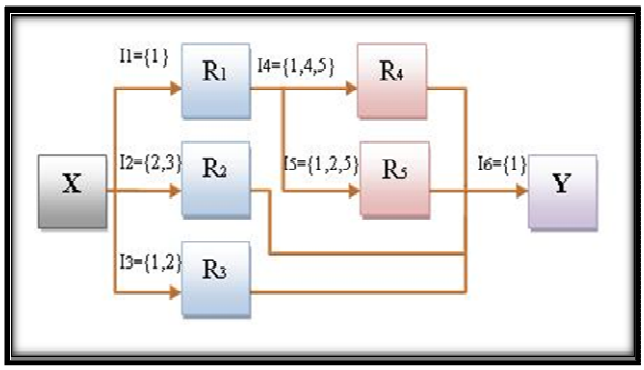

Рис. 2. Граф регистровых передач

Число точек контроля в графе (регистры, переменные, память) должно быть достаточным для постановки диагноза с заданной глубиной. В противном случае необходимо выполнить анализ тестопригодности ГРП программного продукта и определить минимальное дополнительное количество линий наблюдения для формирования ассерций, которые позволят выявлять некорректно работающие модули с заданной глубиной диагностирования. Каждая дуга (см. рис. 2) отмечена совокупностью арифметических операций: {1} – суммирование; {2} – умножение; {3} – вычитание; {4} – деление; {5} – нахождение тригонометрического синуса.

Способ представления алгоритма программы с помощью графовой структуры позволяет наглядно продемонстрировать все возможные варианты выполнения программы, а также облегчает проведение следующего этапа поиска дефекта в программном коде и создании минимального теста.

3.Синтез и анализ тестов

Множество дуг записывается в форме ДНФ вершин, где каждый терм представляет собой одномерный путь от входного порта до выхода, который также покрывает подмножество внутренних линий: $P = X14Y \vee X15Y \vee X2Y \vee X3Y$. В совокупности, одномерные пути, представленные в ДНФ, покрываются все возможные транзакции – вершины и дуги в графе. Каждой дуге ставится в соответствие совокупность фрагментов кода или операторов – активизирующих инструкций, записанных с помощью дизъюнкции. Так, например, путь X14Y активизирует выполнение операций на дугах I_1, I_4, I_6. При этом дуги I_1 и I_6 имеют только по одному оператору, а идентификатору I_4 соответствует последовательное выполнение трёх операций. Тест $P_1 = [(1)(1 \vee 4 \vee 5)(1)]$, активизирующий путь X14Y, обеспечивает проверку корректности выполняемых операций. Таким образом можно записать тест минимального покрытия всех вершин и дуг графа с помощью команд, активизирующих дуги графа, а значит продвижение данных до точек наблюдения. После проведения преобразований получаем: $P = (111 \vee 141 \vee 151) \vee (111 \vee 121 \vee 151) \vee (21 \vee 31) \vee (11 \vee 12)$.

Полученный тест является избыточным, что для программного продукта большой размерности не всегда является приемлемым из-за значительного количества тестовых

наборов. Поэтому важно уметь создавать тест минимальной длины, но с заданной глубиной диагностирования, удовлетворяющей пользователя.

3. Построение таблицы неисправностей

ТН ориентирована на проверку совокупностей фрагментов кодов на дугах, создающих пути активизации данных до точек наблюдения – вершин графа. По результатам сравнения экспериментальных данных тестируемого программного средства и ожидаемых реакций формируется вектор экспериментальной проверки V. В случае несовпадения результатов на наблюдаемой линии соответствующая координата ВЭП принимает единичное значение 1 для рассматриваемого тестового набора. Ниже представлена ТН совокупностей фрагментов кодов на полном тесте $P = X14Y \vee X15Y \vee X2Y \vee X3Y$, где тестовые наборы записаны в общем виде (совокупность одномерных путей).

Глубина диагностирования для такого теста, при значении вектора экспериментальной проверки V = (0100) определяется тремя возможными дефектами: $F = I_{51}, I_{52}, I_{55}$. Единичное значение ВЭП для рассматриваемого тест-вектора означает, что при подаче второго набора, происходит активизация выполнения соответствующих операций. Наименьшее множество термов ДНФ, которое различает все одиночные дефекты программных фрагментов графа регистровых передач, составляет минимальный тест диагностирования. Следующая совокупность термов – здесь совпадает с полным тестом – различает дефекты всех инструкций, заданных в ДНФ: $P = (111 \vee 141 \vee 151) \vee (111 \vee 121 \vee 151) \vee (21 \vee 31) \vee (11 \vee 12)$. Невозможность упрощения обусловлена тем, что исключение любого из термов не обеспечивает активизацию одного или нескольких фрагментов. Далее строится полная и расширенная таблица неисправностей, формируемая с помощью записанной выше совокупности термов. Каждый из полученных тестовых наборов разбивается на составные части – термы. Первый тестовый набор $(111 \vee 141 \vee 151)$ состоит из трёх термов: (111), (141) и (151). Каждый из них занимает свою позицию в столбце. По горизонтали откладываются все возможные операции, выполняемые в программе. Рассматривается путь графа, к которой относится исследуемый терм. Например, терм (141) относится к первому тестовому набору, активизирующему путь X14Y. Расширенная таблица неисправностей имеет вид:

$T_i \setminus I_i$	I_{11}	I_{22}	I_{23}	I_{31}	I_{32}	I_{41}	I_{44}	I_{45}	I_{51}	I_{52}	I_{55}	I_{61}	V
111_1	1					1						1	0
141	1						1					1	0
151_1	1							1				1	0
111_2	1								1			1	1
121	1									1		1	1
151_2	1										1	1	1
21_1		1										1	0
31			1									1	0
11				1								1	0
21_2					1							1	0

Каждая из цифр терма обозначает выполнение одной из операций на соответствующей дуге графа. Первая цифра 1 обеспечивает активацию оператора $\{1\}\ I_1$, поэтому напротив соответствующего столбца ставится единица. Значения столбцов расширенной ТН переносятся из таблицы неисправностей фрагментов кодов, определенной на полном обобщенном тесте. Только теперь значение координаты записывается для каждого из термов теста. Получение расширенной таблицы неисправностей позволяет более наглядно продемонстрировать результаты воздействия

каждого из тестовых наборов, а также упрощает дальнейшее проведение процедуры поиска дефекта для определения местонахождения неисправности с наперед заданной глубиной диагностирования.

4. Определение диагноза.

В соответствии с числом единиц в векторе экспериментальной проверки V формируется количество дизъюнктивных термов КНФ. Каждый терм есть построчная запись дефектов через логическую операцию ИЛИ, оказывающих влияние на искажение выходных сигналов функциональности. Далее осуществляется преобразование КНФ к ДНФ по правилам алгебры логики, и процедура упрощения, что дает следующий результат:

$$F = I_{11} \vee I_{11} I_{55} \vee I_{11} I_{61} \vee I_{11} I_{52} \vee I_{11} I_{52} I_{55} \vee I_{11} I_{52} I_{61} \vee I_{61} I_{11} \vee I_{11} I_{61} I_{51} \vee I_{11} I_{61} \vee I_{51} I_{11} \vee I_{11} I_{51} I_{55} \vee$$

$$\vee I_{11} I_{51} I_{61} \vee I_{11} I_{51} I_{52} \vee I_{51} I_{52} I_{55} \vee I_{51} I_{52} I_{61} \vee I_{51} I_{61} I_{11} \vee I_{55} I_{61} \vee I_{51} I_{61} \vee I_{11} I_{61} \vee I_{11} I_{55} I_{61} \vee I_{11} I_{61} \vee$$

$$\vee I_{11} I_{52} I_{61} \vee I_{52} I_{55} I_{61} \vee I_{52} I_{61} \vee I_{61} I_{11} \vee I_{61} I_{51} \vee I_{61} = I_{11} \vee I_{51} I_{52} I_{55} \vee I_{61}.$$

Далее исключаются такие элементы I_{jk} из F, которые выполняются при использовании других тестовых наборов со значением $V_i = 1$. В результате упрощения остаётся единственный терм ДНФ:

$$F' = F \setminus H = (I_{11} \vee I_{51} I_{52} I_{55} \vee I_{61}) \setminus (I_{11} \vee I_{22} \vee I_{23} \vee I_{31} \vee I_{32} \vee I_{44} \vee I_{45} \vee I_{61}) = I_{51} I_{52} I_{55}.$$

Это означает, что программное средство ошибочно функционирует при выполнении одного из операторов $\{1,2,5\}$ на дуге I_5. Ошибка действительно была совершена в линейном участке программы, принадлежащем дуге с последовательностью операторов I_5, а именно I_{51} – выполнение операции вычитания вместо суммирования.

Получение более точного диагноза (вплоть до оператора) возможно при использовании большего количества контрольных точек, что усложняет поиск дефектов из-за необходимости составления более длинных тестов. Предложенный метод позволяет производить анализ программного средства на наличие ошибок в коде и помогает определить её местоположение. Тестирование и верификация программного продукта является одной из главных проблем при программировании, решение которой повышает качество функционирования программного средства и позволяет избежать непредвиденных результатов работы. Предложенный метод основывается на представлении алгоритма программы с помощью графовой структуры, где дуги есть последовательности операций или фрагменты кодов, а вершины – точки мониторинга информации для составления ассерций. Создание минимального количества тестовых наборов позволяет значительно уменьшить временные затраты на поиск ошибки. При этом тесты должны покрывать все возможные транзакции. Число точек контроля также должно быть минимальным, но достаточным для постановки диагноза с заданной глубиной диагностирования.

5. Выводы

Рассмотрены инновационные технологии тестопригодного проектирования программных продуктов, ориентированные на эффективную разработку тестов и верификацию.

Представлена универсальная модель программного и аппаратного компонента в виде ориентированного графа регистровых передач и управления, на котором можно решать задачи тестопригодного проектирования, синтеза и анализа тестов.

Практическая значимость предложенных методик и моделей заключается в высокой заинтересованности софтверных компаний в инновационных решениях проблемы эффективного тестирования и верификации программных продуктов, предложенных выше.