УДК 004.415

С.А. ВИЛКОМИР

Университет Восточной Каролины, США

АЛГОРИТМ ПОСТРОЕНИЯ МОДЕЛИ ВХОДНОГО ПРОСТРАНСТВА ДЛЯ ТЕСТИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Рассматривается задача построения модели входного пространства программного обеспечения, учитывающей зависимости между параметрами. Данная модель позволяет генерировать все фактически возможные тестовые наборы и может быть использована для статистичесого тестирования программного обеспечения. Предложен алгоритм построения графовой модели для произвольного числа зависимостей, использующий систематическое расщепление подграфов. Детальный пример иллюстрирует предложенный подход.

Ключевые слова: *програмное обеспечение, тестирование, входные параметры, зависимости, генерация тестовых наборов.*

Введение

Задача генерации тестовых наборов для тестирования программного обеспечения с большим пространством значений входных параметров рассматривалась многими исследователями. Предлагались различные комбинаторные подходы [1], в частности, методы «pairwise» [2] и «t-wise» [3] тестирования. Для статистического тестирования применялись модели основанные на цепях Маркова [4, 5].

Проблема возникает, когда между значениями входных параметров существуют зависимости, т.е. не все комбинации входных значений возможны. Для таких ситуаций были предложены некоторые подходы [6, 7], однако в целом эта проблема еще не решена.

Целью данной статьи является построение модели, учитывающей зависимости между входными параметрами и генерирующей только фактически возможные тестовые комбинации. Такая модель необходима для статистичесого тестирования, когда большое число тестов проводится с целью последующей оценки надежности программного обеспечения.

Модель представляет собой направленный граф, в котором вершины ассоциируются с входными параметрами програмного обеспечения и дуги ассоциируются со значениями этих параметров. Данная статья является дальнейшим развитием предыдущих работ [8, 9] и предлагает общий алгоритм построения модели для произвольного числа зависимостей между входными параметрами.

Статья организована следующим образом. В разделе 1 дается формальное определение зависимости между значениями входных параметров. В разделе 2 обсуждаются основные свойства и характеристики модели. Основной результат статьи, алгоритм построения модели для произвольного числа зависимостей, рассмотрен в разделе 3. Раздел 4 содержит детальный пример построения модели для пяти входных параметров и трех зависимостей между ними.

1. Зависимости между параметрами

Рассмотрим пример зависимости между параметрами для программы, работающей с отдельными символами. Параметрами такой программы могут быть *type* и *code*, где *type* описыват тип символа («цифра», «заглавная буква», «строчная буква», «специальный символ») и *code* содержит ASCII код символа от 0 до 127. Тогда примером зависимости является следующее: если *type*=«заглавная буква», то $code \in \{65, \ldots, 90\}$.

Формальное определение зависимости между параметрами x и y с множествами значений X и Y задается разбиением этих множеств на подможества $\{X_1, \ldots, X_t\}$ и $\{Y_1, \ldots, Y_t\}$, такие что:

- Для любых j, m, $1 \le j,m \le t$, $j \ne m$, выполняется $\cup X_j = X$ and $X_j \cap X_m = \varnothing$.
- Для любых j, m, $1 \le j,m \le t$, $j \ne m$, выполняется $\cup Y_j = Y$.

Тогда зависимость определяется взаимнооднозначным соответствием f:

$$f: \{X_1, \ldots, X_t\} \rightarrow \{Y_1, \ldots, Y_t\}.$$

В терминах «если - то» это означает, что если $x \in X_j$ то $y \in f(X_j)$.

В случае такой зависимости, мы будем говорить, что y зависит от x. Данное свойство является симметричным, т.е. x в свою очередь зависит от y (конечно, с другим разбиением множеств значений на подмножества).

2. Модель входного пространства

Проиллюстрируем идею создания модели входного пространства на простом примере двух параметров x и y, каждый из которых может принимать значения от 1 до 4. Если зависимость между x и y отсутствует, модель пространсва входных значеней может быть представлена линейным направленным графом (рис. 1):

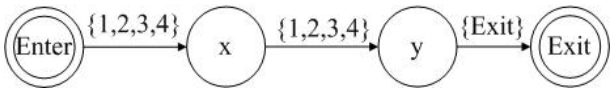

Рис. 1. Модель без зависимостей между параметрами

Каждая дуга графа имеет атрибут - возможные значения параметра, для которого эта дуга является входящей. Выбирая одно значение для каждой дуги, мы тем самым фиксируем значения для каждого параметра и получаем один из возможных тестов.

Допустим теперь, что существует следующая зависимость между параметрами x и y:

если x ∈{1, 2} то y ∈{1, 2, 3 } и

если x ∈{3, 4} то y ∈{2, 3, 4 }.

Для того, чтобы учесть данную зависимость, исходная модель на рис. 1 должна быть преобразована. Преобразование заключается в расщеплении вершины, соответствующей параметру x, на две новых вершины (рис. 2).

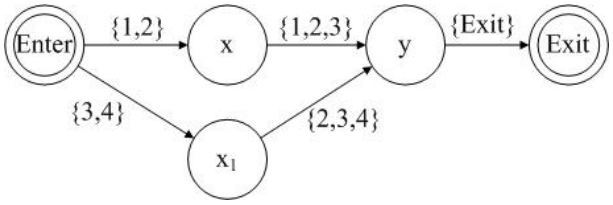

Рис. 2. Модель зависимости между x и y

Данный граф имеет два различных пути от начальной до конечной вершины, при этом каждый путь соответствует множеству тестовых наборов, удовлетворяющих зависимости между x и y.

В раделе 3 рассмотрен алгоритм, который позволяет построить графовую модель для произвольного числа зависимостей. При этом один и тот же параметр может участвовать в нескольких зависимостях одновременно.

3. Алгоритм построения модели

Идея алгоритма построения модели заключается в последовательном преобразовании графа отдельно для каждой зависимости: на основе линейного графа строится граф для первой зависимости, затем на основе графа для первой зависимости строится граф, отражающий первую и вторую зависимости одновременно и т.д. Каждое преобразование имеет три типовых шага (разделы 3.1 – 3.3).

3.1. Расщепление подграфа

Пусть x_1, x_2, …, x_n являются входными параметрами и проводится преобразование для зависимости между x_i и x_j. Рассмотрим подграф, в который входят все вершины, соответствующие параметрам x_i, x_{i+1}, …, x_{j-1} включительно. Произведем расщепление этого подграфа на t идентичных подграфов, где t есть число подмножеств в разбиении множества значений x_i. Все новые подграфы соединяются с предшественниками (вершинами для x_{i-1}) и последующими элементами (вершинами для x_j) точно также, как и исходный подграф.

3.2. Изменение атрибутов ребер

Атрибуты (возможные значения параметров) внутренних ребер подграфов остаются без изменений. Атрибуты входящих и исходящих ребер всех расщепленных подграфов должны быть изменены.

Пусть множество A являлось атрибутом входящего ребра k-того подграфа ($1 \le k \le t$). Тогда его новым атрибутом является $A \cap Xk$, пересечение A и k-того подмножества из разбиения множества значений xi. Аналогично, новым атрибутом исходящего ребра k-того подграфа является $B \cap Yk$, пересечение предыдущего атрибута B и k-того подмножества из разбиения множества значений xj.

3.3 Удаление мертвых ребер и вершин

Следующие ребра и вершины рассматриваются как «мертвые» и должны быть удалены:

- Ребра с атрибутом – пустое множество.
- Вершины без входящих ребер.
- Вершины без исходящих ребер.
- Ребра, соединенные с мертвыми вершинами.

Мертвые ребра могут возникать при изменении атрибутов, когда пересечение множеств является пустым множеством. После удаления таких ребер некоторые вершины могут оказаться без входящих (исходящих) ребер и в свою очередь должны быть удалены. При этом должны быть удалены существующие исходящие (входящие) ребра мертвых вершин. После очередного удаления мертвых ребер могут возникнуть новые мертвые вершины. Таким образом, процесс поиска и удаления мертвых ребер и вершин должен циклично повторяться до тех пор, пока все такие ребра и вершины будут удалены.

4. Пример построения модели

Рассмотрим пример применения предложенного алгоритма для программного обеспечения с пя-

тью входными параметрами u, v, x, y, z, принимающими значения от 1 до 4, и тремя зависимостями между ними:

$$u \in \{1, 2\} \Rightarrow y \in \{1, 2, 3\}; u \in \{3, 4\} \Rightarrow y \in \{1, 2\}.$$
$$v \in \{1, 4\} \Rightarrow z \in \{1, 2\}; v \in \{2, 3\} \Rightarrow z \in \{3, 4\}.$$
$$y \in \{3\} \Rightarrow z \in \{3\}; y \in \{1, 2, 4\} \Rightarrow z \in \{1, 2, 4\}.$$

Построение модели начинается с линейного графа, имеющего семь вершин: пять вершин для входных параметров плюс начальная и конечная вершины.

Для моделирования первой зависимости между u и y, проведем расщепление подграфа с вершинами u, v, и x на два идентичных подграфа. Мертвых ребер и вершин при этом не возникает. Результат показан на рис. 3.

На следующем шаге модель отражает первую и вторую зависимости вместе. Для этого было необходимо провести расщепление подграфа с вершинами v, v_1, x, x_1, и y. Результат показан на рис. 4.

Изменение атрибутов ребер проиллюстрируем на примере ребра (u, v). Начальным атрибутом данного ребра является множество {1..4} (см. рис.2).

Первым подмножеством в разбиении множества значений параметра v является {1,4} (см. выше определение зависимости между v и z). Новым атрибутом является $\{1, 2, 3, 4\} \cap \{1,4\} = \{1,4\}$, что показано на рис. 3. Как и на предыдущем шаге, мертвых ребер и вершин при расщеплении не возникает.

Для моделирования зависимости между y и z производится расщепление подграфа с двумя вершинами y и y_1. При изменении атрибутов ребер некоторые ребра и вершины оказываются мертвыми. Рассмотрим, например, ребро (y, z). Его предыдущий атрибут – множество {1, 2} (рис. 4). Из определения зависимости между y и z, следует что, $Y_1 = \{3\}$ для параметра z. Поэтому новый атрибут ребра определяется как $\{1, 2\} \cap \{3\} = \varnothing$, т.е. ребро (y, z) является мертвым.

Все мертвые ребра и вершины отмечены знаком «**X**» на рис. 5. Окончательная модель всех трех зависимостей между параметрами после удаления мертвых ребер и вершин показана на рис. 6.

Общее количество различных комбинаций значений входных параметров без учета зависимостей равно $4^5 = 1024$. Число фактически возможных тестовых наборов (т.е. допустимых комбинаций значений с учетом зависимостей) равно 208.

Все допустимые (и только допустимые) комбинации могут быть генерированы с помощью графовой модели (рис. 6). Для рассмотренного примера графовая модель имеет 5 различных путей, каждый из которых соответствует определенному множеству тестовых наборов (табл. 1).

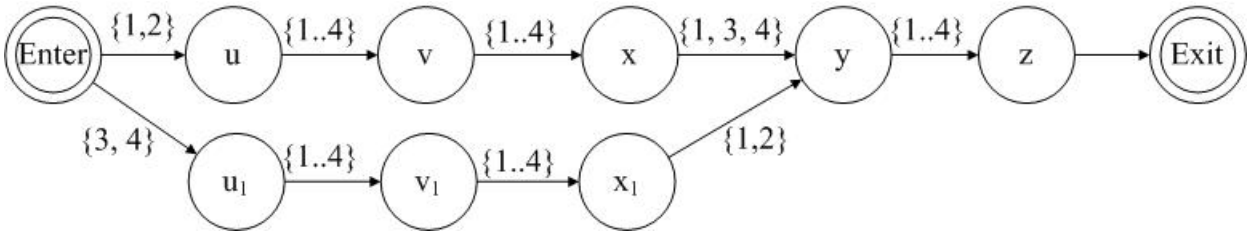

Рис. 3. Модель зависимости между u и y

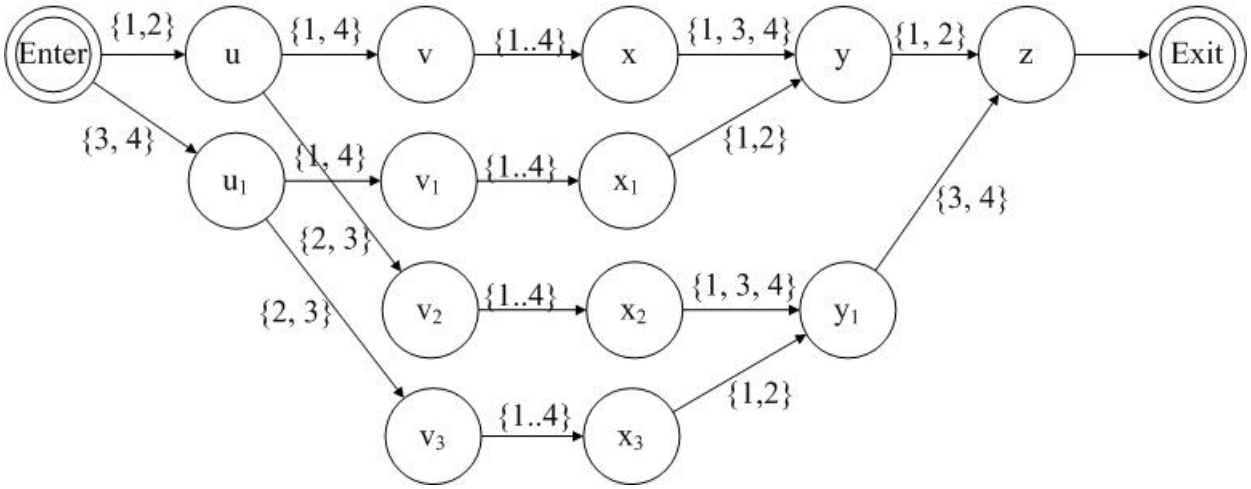

Рис. 4. Модель зависимостей между u и y и между v и z

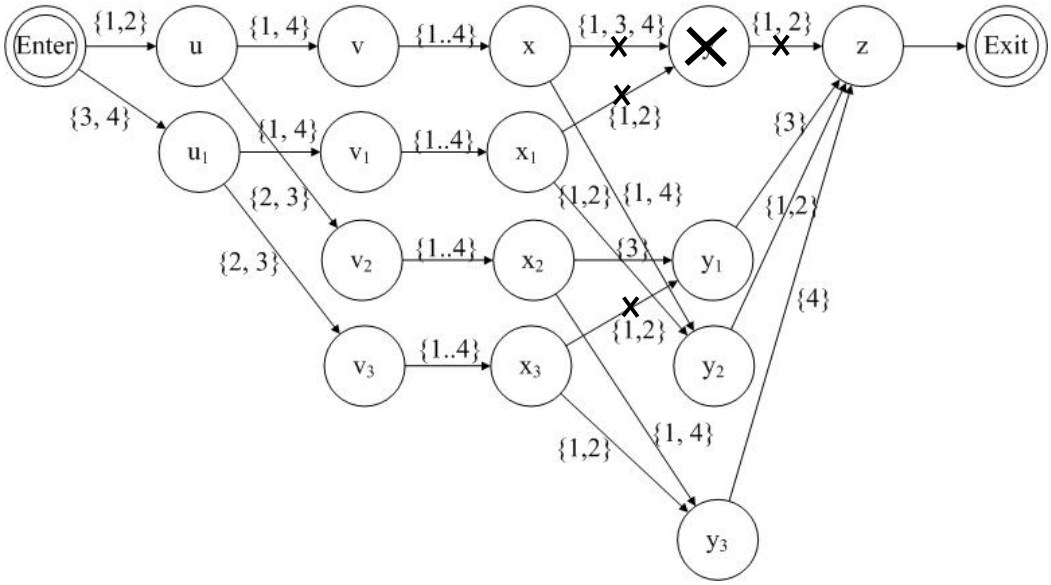

Рис. 5. Удаление мертвых ребер и вершин

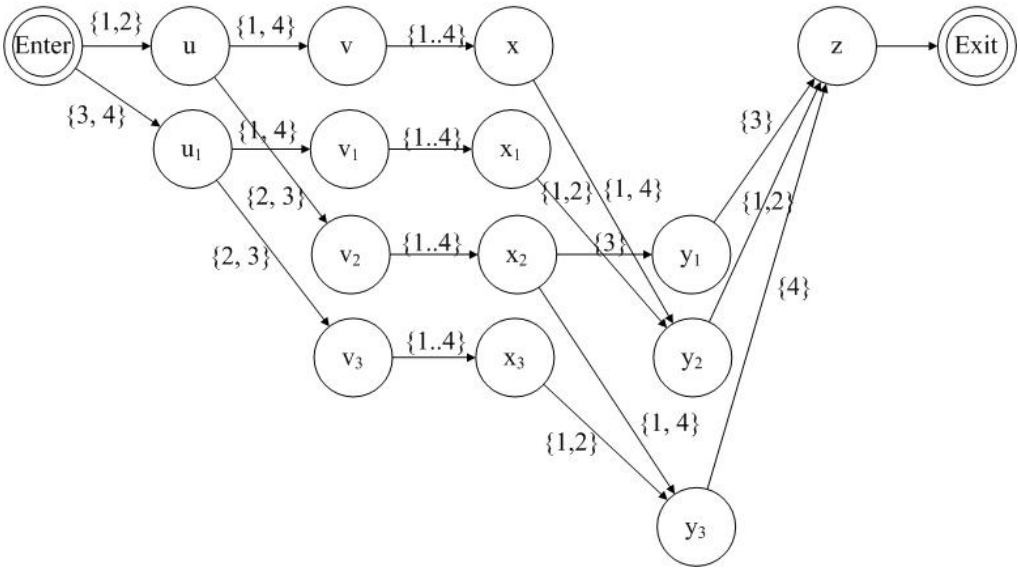

Рис. 6. Окончательная модель зависимостей между параметрами

Таблица 1

Число тестовых наборов

N	Путь	Число тестовых наборов
1	u, v, x, y_2, z	64
2	u, v_2, x_2, y_1, z	16
3	u, v_2, x_2, y_3, z	32
4	u_1, v_1, x_1, y_2, z	64
5	u_1, v_3, x_3, y_3, z	32
Всего		208

К данной модели могут быть добавлены вероятности, с которыми каждый параметр принимает то или иное значение. При этом, становится цепью Маркова, для которой доступен целый ряд программных утилит, анализирующих модели и автоматически генерирующих тестовые наборы. В частности, утилита JUMBL [10] позволяет генерировать тесты случайным образом в соответствии с имеющимся распределением вероятностей, генерировать заданное количество наиболее вероятных входных комбинаций, обеспечивать различные виды покрытий графа и т.п. Комбинация предложенной модели с подобными утилитами дает удобный и эффективный инстумент для тестирования крупных программных систем.

Заключение

Моделирование входного пространства программного обеспечения является первым шагом для автоматической генерации тестовых наборов. Рассмотренный в статье алгоритм строит графическую модель входного пространства, основываясь на зависимостях между входными параметрами. Алгоритм использует расщепление подграфов и может

быть применен при любом количестве зависимостей, рассматривая их последовательно одну за другой. Построенные модели отличаются небольшим размером, при этом позволяя генерировать большое число тестовых наборов. Для автоматической генерации тестовых наборов из графовой модели могут быть использованы существующие программные утилиты, работающие с цепями Маркова [10]. Некоторые результаты применения нами предложенного подхода для тестирования сложных научных вычислительных программ могут быть найдены в [9]. Наиболее целесообразно применение данного подхода для статистического тестирования с последующей оценкой надежности программного обеспечения.

Литература

1. Grindal M. Combination testing strategies: A survey / M. Grindal, J. Offutt, S.F. Andler // Software Testing, Verification, and Reliability. – 2005. – Vol. 15 (3). – P. 167-199.

2. Tai K.C. A test generation strategy for pairwise testing / K.C. Tai, Y. Lei // IEEE Transactions on Software Engineering. – 2002. – 28 (1). – P. 109-111.

3. Williams A.W. A measure for component interaction test coverage / A.W. Williams, R.L. Probert // Proceedings of the ACSI/IEEE International Conference on Computer Systems and Applications (AICCSA 2001), 2001. – P. 304-311.

4. Whittaker J. Markov Analysis of Software Specifications / J. Whittaker, J. Poore // ACM Transactions on Software Engineering and Methodology. – 1993. – Vol. 2(1). – P. 93-106.

5. Whittaker J. Markov chain model for statistical software testing / J. Whittaker, M.G. Thomason // IEEE Transactions on Software Engineering. – 1994. – Vol. 20(10). – P. 812-824.

6. Calvagna A. A Logic-Based Approach to Combinatorial Testing with Constraints / A. Calvagna, A. Gargantini // Chapter in Tests and Proofs, Springer. Lecture Notes in Computer Science. –2008. – Vol. 4966. – P. 66-83.

7. Grindal M. Managing Conflicts When Using Combination Strategies to Test Software / M. Grindal, J. Offutt, J. Mellin // Proceedings of the 18th Australian Software Engineering Conference (ASWEC 2007), Melbourne, 10-13 April 2007. – P. 255-264.

8. Vilkomir S. Combinatorial test case selection with Markovian usage models / S. Vilkomir, W.T. Swain, J. Poore // Proceedings of the 5th International Conference on Information Technology: New Generations (ITNG 2008), Las Vegas , April 7-9 2008. – P. 3-8.

9. Vilkomir S. Modeling input space for testing scientific computational software: a case study / S. Vilkomir, W.T. Swain, J. Poore, K. Clarno // Proceedings of the International Conference on Computational Science (ICCS 2008), Krakow, June 23-25 2008, Part III, LNCS 5103. – P. 291-300.

10. Prowell S. JUMBL: A Tool for Model-Based Statistical Testing / S. Prowell // Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03), Big Islan, January 6-9 2003

Поступила в редакцию 1.02.2009

Рецензент: д-р техн. наук, проф. В.С. Харченко, Национальный аэрокосмический университет им. Н.Е. Жуковского «ХАИ», Харьков.

АЛГОРИТМ ПОБУДОВИ МОДЕЛІ ВХІДНОГО ПРОСТІРУ ДЛЯ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

С.А. Вілкомір

Розглядено задачу побудови моделі вхідного простіру програмного забезпечення, враховуючу залежність між параметрами. Наведена модель дозволяє генерувати усі фактично можливі тестові набори і може бути використована для статистичного тестування програмного забезпечення. Запропонован алгоритм побудови графової моделі для довільної кількості залежностей, використуючий систематичне розщеплювання подграфів. Детальний приклад ілюструє запропонований підхід.

Ключові слова: програмне забезпечення, тестування, вхідні параметри, залежності, генерування тестових наборів.

ALGORITHM FOR CONSTRUCTION OF AN INPUT SPACE MODEL FOR SOFTWARE TESTING

S.A. Vilkomir

A task of construction of software input space model with dependencies among parameters is considered. The model allows generating all possible test cases and can be used for statistical software testing. An algorithm of graph model construction that uses systematic subgraph splitting is proposed. A detailed example illustrates the proposed approach.

Key words: software, testing, input parameters, dependencies, test case generation.

Вилкомир Сергий – доктор философии, отделение компьютерных наук, Университет Восточной Каролины, США, e-mail: vilkomirs@ecu.edu.