Тестирование на основе моделей

В. В. Кулямин

Лекция 5. Комбинаторные методы построения тестов

Комбинаторные методы построения тестов основаны на разделении каждого тестового воздействия на ряд элементов и построении тестов как всевозможных комбинаций полученных элементов, объединяемых по определенным правилам.

Комбинаторные методы дают более высокую полноту покрытия, чем вероятностные, и при этом требуют ненамного больше ресурсов. Кроме того, они хорошо автоматизируются. Однако, с помощью комбинаторных методов трудно найти ошибки в очень специфических ситуациях, требующих учета многих факторов, а трудозатраты на их применение при учете возрастающего числа факторов растут гораздо быстрее.

Одним из примеров комбинаторных методов является *тестирование по разбиениям на категории* (category partition testing). В рамках этого подхода для построения тестов выполняются следующие действия.

- Выделяется набор операций тестируемой системы, обращения к которым должны производиться в тестах.
- Для каждой тестируемой операции анализируются требования к ней и на основе этого анализа выделяются, дополнительно к ее параметрам, некоторые факторы (внешние условия или свойства внутреннего состояния системы), от которых может зависеть ее поведение.
- Возможные значения каждого параметра операции или фактора, влияющего на ее поведения, классифицируются — разбиваются на конечное множество *категорий*. Категории выделяются таким образом, чтобы изменение значения параметра или фактора в рамках одной категории слабо изменяло требования к работе операции.
- Определяются зависимости между полученными категориями, взаимосвязи между значениями различных параметров и факторов, а также недопустимые комбинации их значений.
- Тестовые ситуации строятся как возможные комбинации категорий значений параметров и факторов. Для каждой такой комбинации определяются соответствующие конкретные значения параметров и способ достижения соответствующих значений факторов (тестовые последовательности и изменения внешних условий).

Достаточно сильно похож на эту технику и метод построения тестов *на основе дерева классификации* (classification tree method). Он используется при наличии большого количества факторов, влияющих на поведение тестируемой системы.

Сначала выделяется набор наиболее заметных факторов, влияние которых на поведение тестируемой системы достаточно сильно. Такие факторы называются в рамках этого метода *аспектами*. Для каждого из этих факторов пытаются определить разбиение его возможных значений на группы, в рамках которых требования к поведению системы меняются слабо. При этом могут быть выявлены другие факторы, чье влияние на систему становится существенным, если один из базовых аспектов зафиксирован.

Базовые аспекты образуют вершины дерева классификации, непосредственно связанные с его корнем. Их классы и вторичные аспекты привязываются к базовым аспектам, и т.д., пока все существенные факторы не будут найдены и классифицированы.

После построения дерева классификации необходимо определить зависимости между выделенными классами значений аспектов.

Тесты строятся как возможные комбинации классов значений аспектов, соответствующих листовым вершинам дерева.

Например, при тестировании встроенного программного обеспечения управления автомобилем, возможна следующая классификация ситуаций.

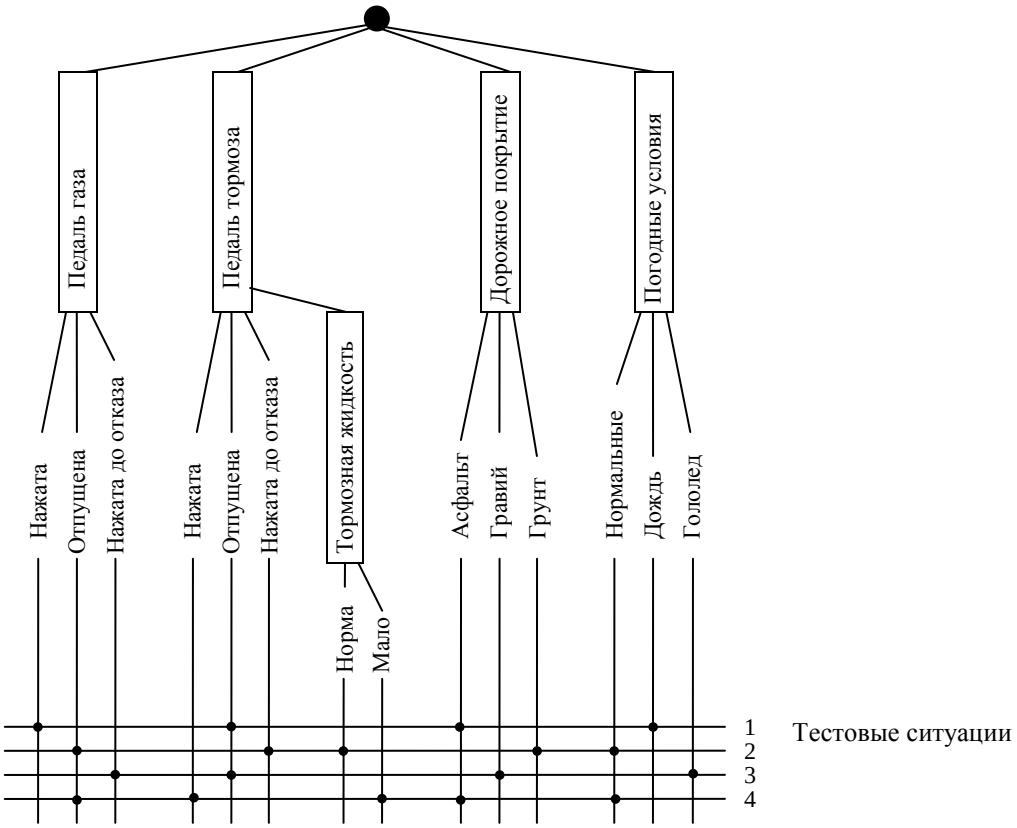

Рисунок 1. Пример использования дерева классификаций.

Тестирование на основе грамматик

Другой часто применяемой комбинаторной техникой создания тестов является *синтаксическое тестирование* или *тестирование на основе грамматик*. Оно используется в тех случаях, когда структуру множества возможных воздействий на тестируемую систему можно описать при помощи формальной грамматики.

Проще всего эту возможность заметить там, где действительно существуют некоторый специальный язык — например, в системах, работающих с языками программирования, языками запросов (SQL и др.), языками для описания структуры документов (XML, HTML и пр.) или с регулярными выражениями. Однако с помощью формальных грамматик можно описывать и структуру пользовательского интерфейса. Наиболее наглядный пример это — структура опций утилит в операционных системах типа UNIX. Чуть менее привычно использование грамматик для описания графических интерфейсов пользователя (GUI).

Рассмотрим для этого простой пример интерфейса поиска фраз и ключевых слов в стандарте POSIX. Этот интерфейс реализован при помощи разделения окна браузера на три области: основной текст, индекс и форма поиска. Форма поиска может иметь два вида, которые представлены на рисунках ниже.

Эта форма в начальном состоянии показана на Рис. 2, при выборе ссылки Word Search ее вид меняется на изображенный на Рис. 3.

Рисунок 2. Форма поиска — основной вид.

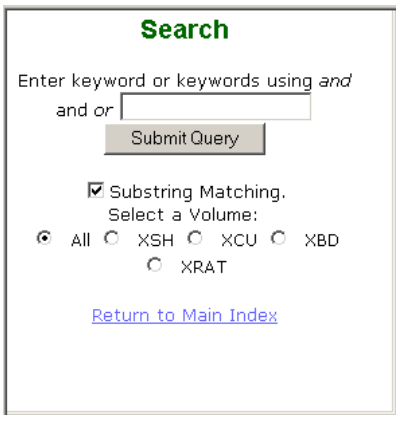

Рисунок 3. Форма поиска ключевых слов.

Все действия пользователя в рамках этой формы поиска можно описать при помощи следующей грамматики.

Chapter ::= Frontmatter | BaseDefinitions | SystemInterfaces | ShellAndUtilities | Rationale

Organization ::= Alphabetical | Topic

MainOption ::= *Chapter* | *Organization*

Section :: = All | XSH | XCU | XBD | XRAT

MainIndexAction ::= (*MainOption*)* (<Phrase> Search)* | *WordSearchAction* ReturnToMainIndex

WordSearchAction ::= *MainIndexAction* WordSearch | *WordSearchAction* (SubstringMatching)? *Section* <Keywords> SubmitQuery

<Keywords> ::= <Word> | <Keywords> "or" <Word> | <Keywords> "and" <Word>

<Phrase> ::= (<Word>)*

В приведенном описании нажатия на кнопки, ссылки или включение/выключение флажков на форме выделены таким шрифтом, как BaseDefinitions или Search. <Keywords>, <Word> и <Phrase> — это различные виды фраз, которые можно ввести в поля ввода формы.

При построении тестов на основе формальных грамматик рассматриваются тестовые ситуации, соответствующие различным вариантам построения предложений в этой грамматике. Похожие наборы вариантов рассматриваются и при определении метрик тестового покрытия на основе грамматик (см. Лекцию 3).

Каждая альтернатива дает столько вариантов, сколько в ней участвует. Опциональный элемент дает два варианта — его присутствие и его отсутствие. Список может давать несколько разных наборов вариантов — можно рассматривать только пустые и непустые списки, а можно рассматривать списки длины 0, 1, 2 и все остальные.

Например, возможные варианты раскрытия правила X ::= (A)? (B | C) D (E)*, можно перечислить так.

ABD CDE ACDEE

Выше перечислены предложения, которые покрывают только каждую из имеющихся альтернатив. Ниже — варианты раскрытия этого же правила, покрывающие все сочетания вариантов раскрытия отдельных альтернатив.

BD BDE BDEE ABD ABDE ABDEE

CD CDE CDEE ACD ACDE ACDEE

Заметим, что здесь возникает много размерностей пространства вариантов, поскольку многие альтернативы могут выбираться независимо от остальных. Поэтому возможны разные стратегии сочетания вариантов — просто раскрыть каждую альтернативу всеми возможными способами, не обращая внимания на их сочетания, перебрать все пары

сочетаний возможных раскрытий соседних альтернатив, перебрать все сочетания возможных раскрытий альтернатив в каждом правиле и пр.

Покрывающие наборы

Покрывающие наборы дают инструмент для систематического перебора различных комбинаций значений параметров или факторов.

Рассмотрим сначала небольшой пример. Предположим, мы тестируем одну операцию, на работу которой может оказать влияние несколько факторов. Примером может быть печать Web-станицы из браузера Интернет, в качестве существенных факторов для которой могут выступать объем печатаемого документа, наличие или отсутствие цветных картинок, размер бумаги, производитель используемого принтера, разновидность используемого браузера, операционная система, которой мы при этом пользуемся.

Чтобы применить комбинирование значений этих факторов, для каждого фактора должно иметься лишь небольшое число различимых значений. Предположим, что, поразмыслив, мы выбрали следующие значения.

Объем	Наличие цветных рисунков	Размер бумаги	Производитель принтера	Браузер	Операционная система
1 страница	Нет цветных рисунков	A4	HP	Internet Explorer	Windows Me
2 страницы	Есть цветные рисунки	A5	Epson	Mozilla Firefox	Windows 2000
7 страниц		B5	Canon	Opera	Windows XP
		Letter	Xerox		Linux SUSE 10.0
		Envelop C5			Linux RHEL 4.0

Таблица 1. Значения параметров для тестирования печати Web-страницы.

Если теперь попробовать составить все возможные комбинации значений факторов, получится 3·2·5·4·3·5 = 1800 тестов. Это не чересчур много, но ясно, что выполнение их всех потребует значительных затрат.

Однако можно существенно сократить эти затраты, если учесть, что подавляющее большинство ошибок в таких ситуациях (до 70%) связано с определенными комбинациями всего лишь двух факторов, то есть, если тесты будут содержать *все возможные комбинации пар значений факторов*, большая часть ошибок будет ими выявлена.

Принятие такого подхода позволяет выполнить лишь небольшое количество тестов, таких, что в них задействованы *все пары значений различных факторов*. В данном примере один из возможных минимальных наборов тестов представлен в Таблице 2.

Показанный набор минимален, поскольку для использования всех сочетаний размеров бумаги и операционных систем необходимо не менее 25 тестов.

Можно пойти еще дальше и попробовать составить тестовый набор так, чтобы он по-прежнему оставался небольшим, но содержал уже *все различные тройки значений факторов*. При этом потребуется не менее 100 = 5·5·4 тестов, что все же существенно меньше, чем 1800 (такой набор из 100 тестов действительно существует).

1	1 страница	Нет цветных рисунков	A4	HP	Internet Explorer	Linux RHEL 4.0
2	1 страница	Нет цветных рисунков	A4	HP	Opera	Windows Me
3	1 страница	Нет цветных рисунков	A5	Epson	Internet Explorer	Windows 2000
4	1 страница	Нет цветных рисунков	Envelop C5	Canon	Internet Explorer	Linux RHEL 4.0
5	1 страница	Есть цветные рисунки	B5	Epson	Opera	Windows XP
6	1 страница	Есть цветные рисунки	Letter	HP	Mozilla Firefox	Windows 2000
7	1 страница	Есть цветные рисунки	Envelop C5	Xerox	Opera	Linux SUSE 10.0
8	2 страницы	Нет цветных рисунков	A5	Xerox	Mozilla Firefox	Linux RHEL 4.0
9	2 страницы	Нет цветных рисунков	Letter	Canon	Opera	Linux SUSE 10.0
10	2 страницы	Нет цветных рисунков	Envelop C5	HP	Mozilla Firefox	Windows XP
11	2 страницы	Есть цветные рисунки	A4	Xerox	Opera	Windows XP
12	2 страницы	Есть цветные рисунки	A5	HP	Opera	Linux SUSE 10.0
13	2 страницы	Есть цветные рисунки	B5	Xerox	Opera	Windows 2000

14	2 страницы	Есть цветные рисунки	Envelop C5	Epson	Internet Explorer	Windows Me
15	7 страниц	Нет цветных рисунков	A4	Canon	Internet Explorer	Windows 2000
16	7 страниц	Нет цветных рисунков	A5	Xerox	Opera	Windows Me
17	7 страниц	Нет цветных рисунков	B5	HP	Internet Explorer	Linux SUSE 10.0
18	7 страниц	Нет цветных рисунков	B5	Canon	Mozilla Firefox	Windows Me
19	7 страниц	Нет цветных рисунков	Letter	HP	Internet Explorer	Linux RHEL 4.0
20	7 страниц	Нет цветных рисунков	Letter	Xerox	Internet Explorer	Windows XP
21	7 страниц	Есть цветные рисунки	A4	Epson	Mozilla Firefox	Linux SUSE 10.0
22	7 страниц	Есть цветные рисунки	A5	Canon	Opera	Windows XP
23	7 страниц	Есть цветные рисунки	B5	Epson	Opera	Linux RHEL 4.0
24	7 страниц	Есть цветные рисунки	Letter	Epson	Opera	Windows Me
25	7 страниц	Есть цветные рисунки	Envelop C5	Epson	Internet Explorer	Windows 2000

Таблица 2. Минимальный тестовый набор для тестирования печати Web-страницы.

Эти примеры обобщаются до понятия *покрывающего набора* глубины t. Нам не важны конкретные значения факторов или параметров, важно только, что они образуют конечное множество. Поэтому можно считать, что если некоторый фактор имеет n возможных значений, ими являются числа от 0 до (n-1). Если заданы конечные наборы значений $\{v_{ij}\}$ для k параметров, i-й параметр может принимать n_i различных значений, то *покрывающим набором* глубины t <= k является любой набор из списков значений всех параметров $\{v_{if(j)}\}$, такой что любая комбинация возможных значений любых t параметров встречается в этом наборе хотя бы один раз.

Пары значений параметров покрываются наборами глубины 2, тройки — наборами глубины 3, и т.д.

	0	0	0	0	0	4
	0	0	0	0	2	0
	0	0	1	1	0	1
	0	0	4	2	0	4
	0	1	2	1	2	2
	0	1	3	0	1	1
	0	1	4	3	2	3
	1	0	1	3	1	4
	1	0	3	2	2	3
	1	0	4	0	1	2
	1	1	0	3	2	2
	1	1	1	0	2	3
	1	1	2	3	2	1
	1	1	4	1	0	0
	2	0	0	2	0	1
	2	0	1	3	2	0
	2	0	2	0	0	3
	2	0	2	2	1	0
	2	0	3	0	0	4
	2	0	3	3	0	2
	2	1	0	1	1	3
	2	1	1	2	2	2
	2	1	2	1	2	4
	2	1	3	1	2	0
	2	1	4	1	0	1

Таблица 3. Покрывающий набор, соответствующий показанному выше тестовому набору.

Множество всех покрывающих наборов глубины t с k параметрами, принимающими n_1, …, n_k значений обозначается CA(t, n_1, …, n_k). Минимальное возможное количество рядов в покрывающем наборе обозначается CAN(t, n_1, …, n_k). Таблица 3 представляет пример набора из CA(2, 3,·2,·5,·4,·3,·5) и показывает, что CAN(2, 3,·2,·5,·4,·3,·5) = 25.

Если все параметры могут принимать одно и то же число значений, т.е. $n_1 = n_2 = \ldots = n_k = n$, соответствующий покрывающий набор называется *однородным*. Множество однородных наборов CA(t, n, …, n) также обозначается CA(t; k, n), соответствующее минимальное число рядов в таком наборе — CAN(t; k, n).

Выгода от использования покрывающих наборов определяется тем фактом, что чаще всего существуют покрывающие наборы небольшой мощности, в которых количество рядов значительно меньше, чем число всех возможных комбинаций значений параметров.

Покрывающие наборы могут эффективно использоваться в ситуациях, в которых выполнены следующие условия.

- Есть некоторый вид воздействий на тестируемую систему, имеющий довольно много параметров или факторов, влияющих на его работу.
- Значения каждого из параметров можно разбить на (небольшое) конечное число классов, таких, что все существенные изменения в поведении системы происходят только из-за изменения класса одного из параметров. Иногда просто каждый параметр может принимать лишь значения из небольшого конечного множества.
- Ошибки в поведении системы возникают в основном за счет сочетания небольшого количества факторов, определяемых значениями используемых параметров.
- Дополнительной информации о зависимости между возможными ошибками и какими-либо другими условиями на значения параметров нет.

В частности, покрывающие наборы могут использоваться для определения комбинаций учитываемых факторов при тестировании на основе разбиения на категории, на основе дерева классификации. Также можно применять покрывающие наборы для снижения количества тестов при построении различных комбинаций альтернатив на основе грамматик.

Техники построения однородных покрывающих наборов

Наиболее хорошо развиты техники построения однородных покрывающих наборов. При количестве значений всех параметров равном 2 есть даже простой алгоритм построения минимального покрывающего набора глубины 2 (см. ниже).

Для однородных наборов глубины 2, в которых число значений параметров равно степени простого числа $n = p^k$ есть метод построения, основанный на арифметике конечных полей. Известно, что для каждой степени простого числа p^k есть конечное поле с таким количеством элементов, называемое полем Галуа $GF(p^k)$. Для $k = 1$, т.е. когда число элементов само является простым, GF(p) изоморфно полю вычетов по модулю p — $\mathbb{Z}_p$.

Рассмотрим таблицу из элементов поля $GF(p^k)$, построенную следующим способом.

Первый столбец состоит из n^2 значений, сгруппированных по n одинаковых значений. Каждую такую группу значений, равных i, будем называть i-м блоком. Первому столбцу присваивается номер ∞.

Второй столбец состоит из n^2 значений, выстроенных так, что в каждом блоке встречаются все возможные n значений. Значение, стоящее во втором столбце обозначим через j. Второму столбцу присвоим номер 0.

Все остальные столбцы, с третьего по (n+1)-й, с номерами m = 1...(n-1) построим так, чтобы в блоке i в j-м ряду стояло значение, получаемое как m*i+j в арифметике $GF(p^k)$.

Построенная так таблица будет покрывающим набором из множества CA(2; n+1, n), если рассматривать каждую ее строку как набор значений n+1 параметров, соответствующих столбцам.

Пример для n = 5.

Поскольку поле GF(5) изоморфно полю вычетов по модулю 5, складывать и умножать числа в обычной целочисленной арифметике, а в конце брать вместо результата его вычет по модулю 5. Получаемый таким способом покрывающий набор представлен ниже.

NN	∞	0	1	2	3	4
	0	0	0	0	0	0
	0	1	1	1	1	1
	0	2	2	2	2	2
	0	3	3	3	3	3
	0	4	4	4	4	4
	1	0	1	2	3	4

	1	1	2	3	4	0
	1	2	3	4	0	1
	1	3	4	0	1	2
	1	4	0	1	2	3
	2	0	2	4	1	3
	2	1	3	0	2	4
	2	2	4	1	3	0
	2	3	0	2	4	1
	2	4	1	3	0	2
	3	0	3	1	4	2
	3	1	4	2	0	3
	3	2	0	3	1	4
	3	3	1	4	2	0
	3	4	2	0	3	1
	4	0	4	3	2	1
	4	1	0	4	3	2
	4	2	1	0	4	3
	4	3	2	1	0	4
	4	4	3	2	1	0

Пример для n = 4.

В GF(4) сложение и умножение устроены иначе, чем по модулю 4. Поэтому сначала приведем таблицы сложения и умножения в поле с 4-мя элементами.

+	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

*	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

Получаемый для n = 4 по описанной конструкции покрывающий набор представлен ниже.

NN	∞	0	1	2	3
	0	0	0	0	0
	0	1	1	1	1
	0	2	2	2	2
	0	3	3	3	3
	1	0	1	2	3
	1	1	0	3	2
	1	2	3	0	1
	1	3	2	1	0
	2	0	2	3	1
	2	1	3	2	0
	2	2	0	1	3
	2	3	1	0	2
	3	0	3	1	2
	3	1	2	0	3
	3	2	1	3	0
	3	3	0	2	1

Таким образом можно строить однородные наборы глубины 2 для (n+1) параметра с n значениями при $n = p^k$. Этим показывается, что $CAN(2; p^k+1, p^k) = p^{2k}$.

Похожая конструкция существует для покрывающих наборов глубины $t > 2$ и $n = p^k > t$. Для этого надо взять таблицу из n+1-го столбца и n^t строк. Каждую строку ее можно сопоставить набору a0, a1, …, at-1 элементов из поля $GF(p^k)$. Столбцы так же обозначаются ∞, 0, 1, ..., n-1. Элемент в определенной строке и определенном столбце вычисляется по следующим правилам.

	∞	0	x
$a_0a_1...a_{t-1}$	a_0	a_{t-1}	$\Sigma a_i x^i$

Здесь снова используются сложение и умножение из поля $GF(p^k)$.

Для глубины 3 и $n = 2^k > t$ можно расширить эту таблицу на еще один столбец — два первых столбца обозначим ∞_1 и ∞_2, остальные, как раньше, — 0, 1, ..., n-1.

	∞_0	∞_1	0	x
$a_0a_1a_2$	a_0	a_1	a_2	$\Sigma a_i x^i$

Таким образом, оказывается, что $CAN(t; p^k+1, p^k) = p^{tk}$ при $t < p^k$, а также $CAN(3; 2^k+2, 2^k) = 2^{3k}$ при $k > 1$.

Пример для n = 3, t = 3.

NN	∞	0	1	2
000	0	0	0	0
001	0	1	1	1
002	0	2	2	2
010	0	0	1	2
011	0	1	2	0
012	0	2	0	1
020	0	0	2	1
021	0	1	0	2
022	0	2	1	0
100	1	0	1	1
101	1	1	2	2
102	1	2	0	0
110	1	0	2	0
111	1	1	0	1
112	1	2	1	2
120	1	0	0	2
121	1	1	1	0
122	1	2	2	1
200	2	0	2	2
201	2	1	0	0
202	2	2	1	1
210	2	0	0	1
211	2	1	1	2
212	2	2	2	0
220	2	0	1	0
221	2	1	2	1
222	2	2	0	2

Представленные выше конструкции позволяют строить однородные покрывающие наборы для небольшого числа параметров (<= n+1), принимающих n значений для n являющегося степенью простого числа.

Посмотрим теперь, как можно строить покрывающие наборы для числа значений, не являющегося степенью простого числа. Оказывается, есть общая конструкция покрывающего набора для числа значений, являющегося произведением чисел значений в уже построенных покрывающих наборов: покрывающие наборы с k параметрами глубины t для n_1 и n_2 значений дают покрывающий набор с k параметрами для глубины t для $n_1 \cdot n_2$.

Для его построения обозначим элементы двух исходных наборов через a_{ij} и b_{lj} — у этих наборов одинаковое число столбцов, и, возможно, разное число строк. В первом наборе участвуют элементы от 0 до $(n_1–1)$, во втором — от 0 до $(n_2–1)$. Любое число от 0 до $(n_1 \cdot n_2–1)$ можно однозначно представить в виде $n_2 \cdot q + r$, где q лежит от 0 до $(n_1–1)$, r — от 0 до $(n_2–1)$. Кроме того, обозначим строки новой таблицы парами индексов строк двух исходных таблиц. Тогда ее элементы могут быть построены по формуле $x_{(i,m)j} = n_2 \cdot a_{ij} + b_{mj}$. Подученная так таблица представляет покрывающий набор с k параметрами для глубины t для $n_1 \cdot n_2$.

Как следствие $CAN(t; k, n_1 \cdot n_2) <= CAN(t; k, n_1) \cdot CAN(t; k, n_2)$.

Пример для $n = 6 = 2 \cdot 3$.

		0	1
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	0

		0	1	2
0	0	0	0	0
1	0	1	1	1
2	0	2	2	2
3	1	0	1	2
4	1	1	2	0
5	1	2	0	1
6	2	0	2	1
7	2	1	0	2
8	2	2	1	0

		0	1
00	0	0	0
01	0	1	1
02	0	2	2
03	1	0	1
04	1	1	2
05	1	2	0
06	2	0	2
07	2	1	0
08	2	2	1
10	0	3	3
11	0	4	4
12	0	5	5
13	1	3	4
14	1	4	5
15	1	5	3
16	2	3	5
17	2	4	3
18	2	5	4
20	3	0	3
21	3	1	4
22	3	2	5
23	4	0	4
24	4	1	5
25	4	2	3
26	5	0	5
27	5	1	3
28	5	2	4
30	3	3	0
31	3	4	1
32	3	5	2
33	4	3	1
34	4	4	2
35	4	5	0
36	5	3	2
37	5	4	0
38	5	2	4

Описанные выше техники позволяют строить покрывающие наборы любой глубины с любыми значениями для небольшого количества параметров. Чтобы увеличить количество параметров, нужно использовать другие подходы.

Во-первых, для глубины 2 и произвольного числа параметров k, принимающих только 2 значения, есть алгоритм, позволяющий достаточно быстро находить минимальный возможный покрывающий набор.

Выберем наименьшее N такое, что выполнено $k \le C_{N-1}^{\lceil N/2 \rceil}$. Здесь $\lceil x \rceil$ — наименьшее целое число, большее или равное x, C_r^q — биномиальный коэффициент. Получаемые значения N для разных k сведены в следующую таблицу.

k	N		k	N		k	N
2-3	4		1717-3003	15		2496145-5200300	26
4	5		3004-6435	16		5200301-9657700	27
5-10	6		6436-11440	17		9657701-20058300	28
11-15	7		11441-24310	18		20058301-37442160	29
16-35	8		24311-43758	19		37442161-77558760	30
36-56	9		43759-92378	20		77558761-145422675	31
57-126	10		92377-167960	21		145422676-300540195	32
127-210	11		167961-352716	22		300540196-565722720	33
211-462	12		352717-646646	23		565722721-1166803110	34

463-792	13		646647-1352078	24		1166803111-2203961430	35
793-1716	14		1352079-2496144	25		2203961431-4537567650	36

Это число N равно числу строк в покрывающем наборе глубины 2 с двумя значениями для k параметров. Из таблицы видно, что небольшое число тестов может покрыть все комбинации пар значений для огромного количества параметров — 10 тестов достаточно для 126 параметров, а 20 тестов — для более чем 92000.

Первую строка набора сделаем состоящей целиком из 0. Остается N–1 строк, элементы которых строятся по столбцам. В качестве этих столбцов берутся все возможные последовательности из $\lceil N/2 \rceil$ единиц и $\lfloor N/2 \rfloor$-1 нулей.

Примеры.

CAN(2; 4, 2) = 5

0000
1110
1101
1011
0111

CAN(2; 10, 2) = 6

00000 00000
11111 10000
11100 01110
10011 01101
01010 11011
00101 10111

CAN(2; 15, 2) = 7

00000 00000 00000
11111 11111 00000
11111 10000 11110
11100 01110 11101
10011 01101 11011
01010 11011 10111
00101 10111 01111

CAN(2; 35, 2) = 8

00000 00000 00000 00000 00000 00000 00000
11111 11111 11111 11111 00000 00000 00000
11111 11111 00000 00000 11111 11111 00000
11110 00000 11111 10000 11111 10000 11110
10001 11000 11100 01110 11100 01110 11101
01001 00110 10011 01101 10011 01101 11011
00100 10101 01010 11011 01010 11011 10111
00010 01011 00101 10111 00101 10111 01111

CAN(2; 56, 2) = 9

00000 00000 00000 00000 00000 00000 00000 00000 00000 00000 00000 0
11111 11111 11111 11111 11111 11111 11111 00000 00000 00000 00000 0
11111 11111 11111 11111 00000 00000 00000 11111 11111 11111 00000 0
11111 11111 00000 00000 11111 11111 00000 11111 11111 00000 11111 0
11110 00000 11111 10000 11111 10000 11110 11111 10000 11110 11110 1
10001 11000 11100 01110 11100 01110 11101 11100 01110 11101 11101 1
01001 00110 10011 01101 10011 01101 11011 10011 01101 11011 11011 1
00100 10101 01010 11011 01010 11011 10111 01010 11011 10111 10111 1
00010 01011 00101 10111 00101 10111 01111 00101 10111 01111 01111 1

Доказать, что получаемый так набор действительно покрывающий достаточно просто, а вот для доказательства того, что он минимальный нужны нетривиальные комбинаторные факты, а именно — теорема Ердеша-Ко-Радо.

Для всех остальных значений параметров хороших алгоритмов построения минимальных покрывающих наборов неизвестно, более того, показано, что построение минимальных покрывающих наборов CA(2, k, n), CA(t, k, 2) — NP-полные задачи.

Для построения однородных покрывающих наборов для числа значений, не равного 2, и для большого количества параметров проще всего использовать рекурсивные конструкции, с помощью которых набор для большого количества параметров строится из наборов для меньшего количества. Ниже рассматриваются две такие конструкции — для глубины 2 и для глубины 3.

Рекурсивная конструкция для покрывающих наборов глубины 2 и $n = p^k$.

Строим набор для nm+1 параметра из набора для m параметров.

Обозначим через A_{ij} (i <= N, j <= m) элементы исходного набора из CA(2; m, n).

Обозначим также через B_{ij} элементы из нижней части (без n верхних строк) покрывающего набора, построенного с помощью самой первой конструкции, число строк в наборе B равно $z = n^2–n$.

Строим новый набор в соответствии с приведенной ниже схемой — в ней первая строка, а также первый столбец отмечают группировку элементов.

		n				n					n			
N	0	A_{11}	A_{11}	…	A_{11}	A_{12}	A_{12}	…	A_{12}	…	A_{1m}	A_{1m}	…	A_{1m}
	0	A_{21}	A_{21}		A_{21}	A_{22}	A_{22}		A_{22}		A_{2m}	A_{2m}		A_{2m}
	…	…	…		…	…	…		…		…	…		…
	0	A_{N1}	A_{N1}		A_{N1}	A_{N2}	A_{N2}		A_{N2}		A_{Nm}	A_{Nm}		A_{Nm}
	B_{11}	B_{12}	B_{13}	…	$B_{1(n+1)}$	B_{12}	B_{13}	…	$B_{1(n+1)}$	…	B_{12}	B_{13}	…	$B_{1(n+1)}$
	…	…	…		…	…	…		…		…	…		…
	B_{z1}	B_{z2}	B_{z3}		$B_{z(n+1)}$	B_{z2}	B_{z3}		$B_{z(n+1)}$		B_{z2}	B_{z3}		$B_{z(n+1)}$

Получаем соотношение $CAN(2; mp^k+1, p^k) <= CAN(2; m, p^k) + p^{2k} – p^k$.

Пример для n = 3, m = 4 Исходный набор A, набор B получается из него отбрасыванием 3-х верхних строк.

		0	1	2
0	0	0	0	0
1	0	1	1	1
2	0	2	2	2
3	1	0	1	2
4	1	1	2	0
5	1	2	0	1
6	2	0	2	1
7	2	1	0	2
8	2	2	1	0

1	0	1	2
1	1	2	0
1	2	0	1
2	0	2	1
2	1	0	2
2	2	1	0

Получаемый набор размера 15 для 13 параметров выглядит так.

0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1	1	1	1	1	1
2	0	0	0	0	2	2	2	2	2	2	2	2	2
3	0	1	1	1	0	0	0	1	1	1	2	2	2
4	0	1	1	1	1	1	1	2	2	2	0	0	0
5	0	1	1	1	2	2	2	0	0	0	1	1	1
6	0	2	2	2	0	0	0	2	2	2	1	1	1
7	0	2	2	2	1	1	1	0	0	0	2	2	2
8	0	2	2	2	2	2	2	1	1	1	0	0	0
9	1	0	1	2	0	1	2	0	1	2	0	1	2
10	1	1	2	0	1	2	0	1	2	0	1	2	0
11	1	2	0	1	2	0	1	2	0	1	2	0	1
12	2	0	2	1	0	2	1	0	2	1	0	2	1
13	2	1	0	2	1	0	2	1	0	2	1	0	2
14	2	2	1	0	2	1	0	2	1	0	2	1	0

Рекурсивная конструкция для покрывающих наборов глубины 3.

Строим набор глубины 3 для числа элементов n и 2k параметров из набора глубины 3 для того же числа элементов и k параметров и набора глубины 2 для того же числа элементов и k параметров.

Обозначим элементы исходного набора глубины 3 через A_{ij} (i <= N, j <= k), элементы исходного набора глубины 2 — через B_{ij} (i <= z, j <= k).

Строим новый набор в соответствии с приведенной ниже схемой — в ней первый столбец отмечает группировку элементов, а все сложения проводятся по модулю n.

N	A_{11}	A_{11}	A_{12}	A_{12}	A_{13}	A_{13}		A_{1k}	A_{1k}
	…	…	…	…	…	…		…	…
	A_{N1}	A_{N1}	A_{N2}	A_{N2}	A_{N3}	A_{N3}		A_{Nk}	A_{Nk}
1	B_{11}	$(B_{11}+1)$	B_{12}	$(B_{12}+1)$	B_{13}	$(B_{13}+1)$		B_{1k}	$(B_{1k}+1)$
	…	…	…	…	…	…		…	…
	B_{z1}	$(B_{z1}+1)$	B_{z2}	$(B_{z2}+1)$	B_{z3}	$(B_{z3}+1)$		B_{zk}	$(B_{zk}+1)$
2	B_{11}	$(B_{11}+2)$	B_{12}	$(B_{12}+2)$	B_{13}	$(B_{13}+2)$		B_{1k}	$(B_{1k}+2)$
	…	…	…	…	…	…		…	…
	B_{z1}	$(B_{z1}+2)$	B_{z2}	$(B_{z2}+2)$	B_{z3}	$(B_{z3}+2)$		B_{zk}	$(B_{zk}+2)$
…	…	…	…	…	…	…		…	…
n-1	B_{11}	$(B_{11}+(n-1))$	B_{12}	$(B_{12}+(n-1))$	B_{13}	$(B_{13}+(n-1))$		B_{1k}	$(B_{1k}+(n-1))$
	…	…	…	…	…	…		…	…
	B_{z1}	$(B_{z1}+(n-1))$	B_{z2}	$(B_{z2}+(n-1))$	B_{z3}	$(B_{z3}+(n-1))$		B_{zk}	$(B_{zk}+(n-1))$

Получаем соотношение CAN(3; 2k, n) <= CAN(3; k, n) + (n-1)CAN(2; k, n).

Для представленных выше в различных местах наборов с n = 3 и k = 4 можно получить набор из CA(3; 8, 3), имеющий 45 строк.

Приведенные выше конструкции позволяют строить наборы, число тестов в которых примерно пропорционально логарифму числа параметров. Доказаны следующие соотношения.

При $n\to\infty$ CAN(2, k, n) ~ (n/2)log(k)

CAN(t, k, 2) <~ $t2^t$log(k)O(log(t))

CAN(t, k, n) <~ (t-1)log(k)/log(n^t/(n^t-1)), что при $n^t\to\infty$ можно ограничить выражением $(1+\varepsilon)tn^t$log(k) для любой положительной константы ε.

Построение неоднородных покрывающих наборов

Неоднородные покрывающие наборы часто можно достаточно быстро построить из близких по конфигурации однородных.

Например, самый первый приведенный выше набор из CA(2, 3,·2,·5,·4,·3,·5) был построен так. Переставляя параметры его можно свести к CA(2, 5, 5, 4, 3, 3, 2). Уже видно, что потребуется как минимум 25 строк. Есть однородный набор CA(2, 5, 5, 5, 5, 5, 5), состоящий в точности из 25 строк — берем его и приводим все параметры с меньшим числом значений по модулю этого числа значений.

Описанный прием может не дать покрывающий набор, если есть параметры с числом значений, не взаимно простым с числом значений в исходном однородном наборе. В таких случаях часто все равно можно построить соответствующий неоднородный набор, только для «проблемных» параметров придется отдельно подбирать расположение их значений.

Еще один пример — покрывающий набор из CA(2, 10, 9, 8, 7, 6, 5, 4, 3, 2). Для него требуется не меньше 90 строк, однако близких однородных наборов с 10 значениями нет. Первые два столбца построим просто как все возможные сочетания 10 и 9 элементов. А дальше можно использовать столбцы, начиная с третьего, из однородного набора для 9

элементов из CA(2; 10, 9). Для параметров, число значений которых равно 6 и 3, придется дополнительно подбирать расстановку, но для этого достаточно несколько раз сместить вдоль столбца соответствующие значения из однородного набора, взятые по модулю 6 и 3. В итоге получается искомый покрывающий набор из 90 элементов, а значит — минимальный возможный.

В общем случае для построения покрывающих наборов любой конфигурации, однородных и неоднородных, можно использовать эвристические алгоритмы. Одна из возможных эвристик при построении набора из CA(t, n_1, …, n_k) — построить сначала все возможные комбинации из t значений параметров, а затем последовательно пытаться добавить новые столбцы, добавляя строки только при необходимости, пока не получим искомый набор. Такой алгоритм называется *жадным* и выглядит примерно так.

Упорядочим n_1, …, n_k по убыванию (IPO) и построим исходную таблицу из всех возможных комбинаций значений первых t параметров.

1. Если неиспользованных параметров нет — выдаем построенный набор в качестве результата.
   Если есть еще неиспользованные параметры, берем первый из них. Строим столбец его значений произвольным образом.
2. Вычисляем покрываемые комбинации из значений t уже имеющихся параметров, в которых последним параметром является только что добавленный. Если можно расширить это множество за счет замены значений в только что построенном столбце, делаем это.
   Если покрыты все возможные комбинации значений t параметров с использованием добавленного, идем в пункт 1.
   Иначе идем в пункт 3.
3. Построим новую строку с использованием любой из оставшихся непокрытыми комбинаций значений t параметров с использованием добавленного псоледним. Если в ней есть неопределенные значения параметров, определяем их так, чтобы при этом покрылось как можно больше других непокрытых комбинаций.
   Добавляем построенную строку к таблице и выбрасываем из множества непокрытых комбинаций все, покрываемые ею.
   Если больше нет непокрытых комбинаций, идем в пункт 1.
   Иначе возвращаемся в начало пункта 3.

Этот и другие эвристические алгоритмы построения покрывающих наборов реализованы в множестве инструментов, как коммерческих, так и доступных свободно, таких, как AETG, TestCover.com, CaseMaker, Pro-test, CATS, IBM Intelligent Test Case Handler, TConfig, TCG Jenny (см. [1]).

Комбинаторные методы построения тестовых последовательностей

Рассмотренные выше методы были, в основном, ориентированы на построение тестовых данных. Их можно использовать и для разработки более сложных тестов, вводя элементы состояния как дополнительные факторы. Однако при этом надо отдельно заботиться о преобразовании полученных комбинаторных схем в исполнимые тесты. То есть, если построенный тест определяет, что некоторый элемент состояния должен принадлежать классу X, разработчик тестов должен сам придумать способ установить этот элемент в нужное значение.

Существуют и более прямые комбинаторные методы построения тестовых последовательностей, позволяющих покрывать различные состояния тестируемой системы.

Наиболее простой такой подход основан на последовательностях де Бройна.

Предположим, что в тестируемой системе n операций без параметров, каждая из которых может изменять ее состояние. Каким образом можно построить одну последовательность их

вызовов так, чтобы при этом проверить как можно больше различных вариантов поведения системы?

Один из возможных ответов — использовать *последовательность де Бройна из n значений глубины k.* Это максимально короткая последовательность, которая содержит все возможные последовательности длины k из n значений в качестве своих подпоследовательностей. Известно, что для любых n и k такие последовательности существуют и имеют длину $n^{(k-1)}+(k-1)$. При этом все их подпоследовательности длины k различны, то есть все возможные последовательности длины k упакованы в такую последовательность максимально плотным образом.

Примеры последовательностей де Бройна.

n = 2, k = 2 — 00110

n = 2, k = 3 — 0001011100

n = 2, k = 4 — 0000100110101111000

n = 3, k = 2 — 0010211220

n = 3, k = 3 — 000101110020121120222102122 00

n = 4, k = 2 — 00102031121322330

Для построения последовательностей де Бройна используют графы де Бройна. *Граф де Бройна* с параметрами n≥1 и k≥1 B(m, k) — это ориентированный граф с n^{k-1} вершинами V(n, k) = $[0..(n-1)]^{k-1}$, являющимися всеми возможными последовательностями из n значений длины (k–1), и ребрами E(n, k) = $[0..(n-1)]^{k}$, являющимися всеми возможными последовательностями из n значений длины k. При этом ребро $x_1x_2…x_{k-1}x_k$ начинается в вершине $x_1x_2…x_{k-1}$ и заканчивается в вершине $x_2…x_{k-1}x_k$.

Примеры графов де Бройна изображены на рисунке ниже.

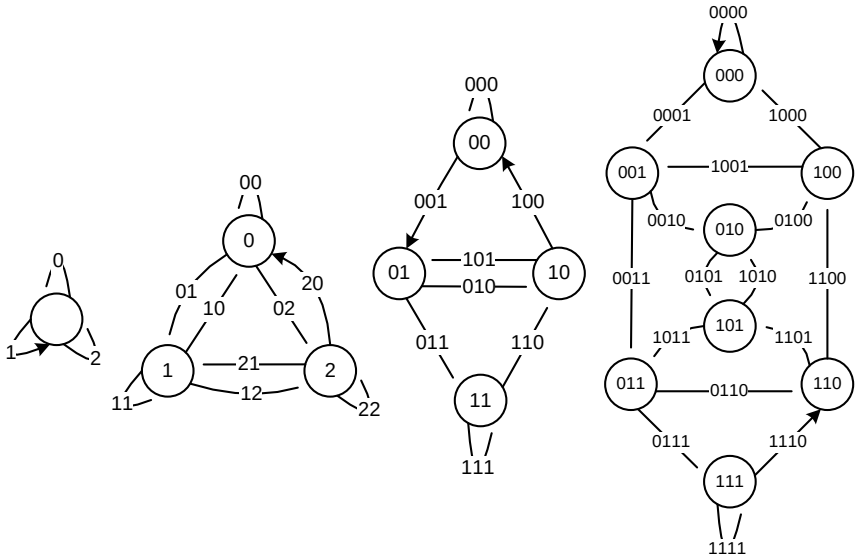

Рисунок 4. Графы B(3, 1), B(3, 2), B(2, 3), B(2, 4).

Все графы де Бройна — эйлеровы, то есть в них существует путь, содержащий все ребра по одному разу, с совпадающими началом и концом. Если выписать последовательность, являющуюся первым ребром такого пути, а для каждого следующего ребра добавлять к полученной последовательности последний символ (заметим, что начало длины k–1 этого ребра будет концом получаемой перед ним последовательсноти), получится как раз последовательность де Бройна.

Каждая последовательность де Бройна глубины k для n значений полностью покрывает все возможные ситуации в тестируемой системе с n операциями, если предположить, что ее поведение полностью определяется последними k вызванными операциями.

Литература

[1] http://www.pairwise.org/tools.asp