Комбинаторная генерация программных конфигураций ОС[1]

Кулямин В. В.
kuliamin@ispras.ru

Аннотация. В статье представлен метод генерации тестов для конфигурационного тестирования на основе покрывающих наборов, т.е., обеспечивающая покрытие всех возможных комбинаций пар, троек и т.д., значений параметров конфигурации. Новым элементом в предлагаемом методе является учет условий использования отдельных параметров, который вносит коррективы как в учет покрываемых комбинаций, так и в построение отдельных тестов. Данный метод использован на практике для генерации тестовых программных конфигураций операционной системы реального времени, приведены результаты этого применения.

Ключевые слова: конфигурационное тестирование; покрывающий набор; генерация тестов

1. Введение

Одной из важных задач контроля качества системного программного обеспечения (ПО) является проверка корректности его работы в разнообразных конфигурациях. Часть вариативности этих конфигураций может быть связана с используемым аппаратным обеспечением, другая часть — чисто программная и представляет собой набор возможных изменений в составе модулей ПО, в настройках используемых алгоритмов, в значениях констант, определяющих, например, емкость внутренних буферов или количество внешних каналов связи определенного типа. Тестирование корректности работы ПО в разных его конфигурациях называется обычно конфигурационным тестированием.

Основная проблема конфигурационного тестирования — необходимость выбора сравнительно малого числа конфигураций для тестирования, которое нужно провести в рамках определенных временных и бюджетных ограничений, из огромного набора всех возможных конфигураций, полная проверка которого просто физически невозможна. Количество параметров

[1]Работа поддержана ФЦП “Исследования и разработ-,ки по приоритетным направлениям развития научно-технологического комплекса России на 2007–2013 годы” (контракт № 11.519.11.4024).

конфигурации практически используемых систем достигает сотен и тысяч, многие из этих параметров могут иметь больше двух значений. Даже для сотни параметров с двумя значениями общее число возможных конфигураций равно $2^{100} = 1.26765\ldots\cdot10^{30}$, для реальных систем это число гораздо больше.

Дополнительные сложности в процесс выбора небольшого набора тестовых конфигураций из огромного множества возможных привносят разнообразные ограничения на корректные наборы значений параметров конфигурации: некоторые параметры могут принимать лишь подмножество возможных значений, если другие зафиксированы (подобные ограничения можно назвать *общими*), часть параметров вообще не используется и не влияет на работу системы, если определенный параметр имеет неподходящее значение (такого рода ограничения будем называть далее *условиями использования*). Например, неважно, куда именно выводится трасса, если вывод трассы отключен, а если он включен, трасса может быть направлена либо на терминал, либо в файл. Ограничения не позволяют использовать в тестовых конфигурациях произвольные значения параметров — они должны соответствовать всем налагаемым ограничениям, при этом выбор тестовых конфигураций усложняется, а сопутствующее уменьшение множества всех возможных конфигураций не упрощает задачи, поскольку его мощность остается слишком большой для любых переборных методов.

Основная цель конфигурационного тестирования в случае программных конфигураций — добиться выполнения различных частей кода системы, которые включаются в нее или исключаются в зависимости от значений конфигурационных параметров (очень часто эти значения задаются через макроопределения #define/#undef, а части кода включаются или исключаются за счет использования #ifdef). Каждый такой участок кода, условно включаемый в код системы, закрыт лишь небольшим количеством условий, обычно одним-двумя, реже используются 3-4. Для его включения достаточно использовать любую конфигурацию, где это небольшое число параметров, от которых он зависит, имеет заданные значения. Поэтому решение проблемы выбора тестовых конфигураций может быть получено за счет применения покрывающих наборов [1], обеспечивающих использование сочетаний всех возможных пар, троек, и т.д. значений конфигурационных параметров в рамках тестов. Тестовые наборы при этом достаточно компактны — размер покрывающего набора растет логарифмически от числа параметров, в то время как общий размер множества конфигураций растет экспоненциально. Кроме того, есть и эмпирические подтверждения того, что покрывающие наборы достаточно эффективны для выявления дефектов при использовании небольших тестовых наборов [1].

Данная работа представляет один из возможных подходов к построению покрывающих наборов, используемых далее для конфигурационного тестирования. Описываемая техника построения покрывающих наборов является эвристической комбинацией широко известных методов [2],

эффективность которой обуславливается особенностями области применения, — в общей ситуации использование этой техники, скорее всего, не будет настолько успешным. Новым элементом в представленной технике является учет условий использования параметров конфигураций (т.е., ограничений на значения других параметров, при нарушении которых значение рассматриваемого параметра вообще не влияет на работу системы), — в литературе и инструментах генерации покрывающих наборов, известных автору, иногда учитываются только общие ограничения на значения параметров, но не условия использования. В последних разделах статьи приводятся результаты практического применения описываемой техники для генерации тестовых программных (не включающих используемого оборудования) конфигураций операционной системы (ОС) реального времени, разрабатываемой в России.

Структура статьи такова: следующий раздел посвящен точной постановке задачи построения тестовых конфигураций, в четвертом разделе описывается применяемый метод решения этой задачи, пятый раздел представляет практические результаты ее использования, статья завершается заключением.

2. Построение тестовых конфигураций на базе покрывающих наборов

Входные данные для построения тестовых конфигураций задаются *классом конфигураций*, который включает следующие элементы.

- Набор *параметров конфигурации*.
  Для каждого параметра указываются
  - Имя параметра.
  - Конечное множество *допустимых значений* этого параметра.
  - *Условие использования* этого параметра. Это логическое ограничение на значения других параметров, при нарушении которого значение данного параметра не используется тестируемой системой. Т.е., такое ограничение может нарушаться в каких-то из тестовых конфигураций, но при этом значение данного параметра не учитывается в качестве значимого.
- *Общие ограничения* на значения параметров. Это логические ограничения на значения параметров, которые должны быть выполнены во всех тестовых конфигурациях.

Условия использования и общие ограничения можно рассматривать как выражения, построенные при помощи логических операций (отрицание, конъюнкция, дизъюнкция) из элементарных выражений вида $p = v$, где p — имя параметра данного класса, v — допустимое значение данного параметра.

(*Допустимой*) *конфигурацией* данного класса считается любой набор значений множества параметров этого класса (отображение имен параметров в их значения), присваивающий каждому параметру значение из множества его допустимых значений и удовлетворяющий общим ограничениям.

Сочетанием или *комбинацией значений* $v_{p_1}, \ldots, v_{p_k}$ параметров $p_1, \ldots, p_k$ будем называть отображение, для каждого i сопоставляющее p_i значение v_{p_i}.

Некоторое сочетание значений $v_{p_1}, \ldots, v_{p_k}$ называется *достижимым* в данном классе конфигураций, если существует допустимая конфигурация этого класса, в которой указанные параметры имеют заданные значения ($p_i = v_{p_i}$) и для каждого параметра p_i выполнено его условие использования. Не каждое сочетание значений является достижимым в произвольном классе конфигураций — некоторые могут нарушать общие ограничения класса, другие могут входить в противоречие с условиями использования участвующих в них параметров.

В данной статье рассматривается задача построения набора тестовых конфигураций на основе покрывающих наборов. Ее можно сформулировать так: для заданного класса конфигураций и натурального числа t построить такой набор допустимых конфигураций этого класса, что для любого достижимого в данном классе сочетания значений любых t различных параметров в построенном наборе есть конфигурация, в которой это сочетание значений реализуется (т.е., данные t параметров имеют заданные значения) и условия использования этих t параметров выполнены.

Связь этой задачи с покрывающими наборами видна из определения последнего понятия. *Покрывающим набором* глубины t для k факторов, могущих принимать, соответственно, $n_1, \ldots, n_k$ значений называется матрица A размера $N \times k$, такая, что в i-ом ее столбце используются числа от 0 до n_i-1 и любая комбинация возможных значений любых t факторов встречается в хотя бы в одной из ее строк, т.е., $\forall j_1, \ldots, j_t \in [1..k]\ \forall v_1 \in [0..n_{j_1}-1], \ldots, v_t \in [0..n_{j_t}-1]\ \exists i \in [1..N]\ \forall q \in [1..t]\ A_{ij_q} = v_q$.

Если в заданном классе конфигураций k параметров, могущих принимать $n_1, \ldots, n_k$ значений, то всякое решение поставленной задачи будет соответствовать покрывающему набору — достаточно, как-то пронумеровав значения каждого параметра от 0 до n_i-1, заменить их на числа. Однако не всякий покрывающий набор дает нужное решение — все строки должны удовлетворять общим ограничениям, а учет комбинаций значений параметров ведется только по тем строкам, которые удовлетворяют условиям использования этих параметров.

Если предположить, что условия использования и общие ограничения не снижают существенно размерность пространства допустимых конфигураций, то можно воспользоваться известными методами [2] для быстрого построения подходящего покрывающего набора, а затем за счет небольших его

трансформаций добиться выполнения всех нужных ограничений. В статье описан именно такой подход к решению данной задачи при отсутствии общих ограничений (при решении ее на практике — построении тестовых конфигураций для операционной системы — таких ограничений не оказалось).

Если же имеющиеся ограничения существенно сокращают набор допустимых конфигураций (так, что, например, значения значительной части параметров можно вычислить, исходя из ограничений и значений оставшейся части параметров), указанная выше стратегия, скорее всего, не будет эффективной. В такой ситуации, по-видимому, проще сначала сократить число параметров, выбросив большинство вычислимых параметров из класса конфигураций.

3. Предлагаемый метод генерации тестовых конфигураций

В данном разделе описывается метод построения набора тестовых конфигураций для заданного класса конфигураций, решающий сформулированную выше задачу при отсутствии общих ограничений.

В качестве исходных данных используются класс конфигураций и натуральное число t — размер комбинаций значений параметров, которые нужно покрыть (для пар значений t = 2? lzk троек t = 3, и т.д.). Построение искомого набора конфигураций выполняется в три этапа.

1. Построение базового набора. Строится покрывающий набор глубины t для k факторов, могущих принимать n_1, …, n_k значений. Здесь k — число параметров в заданном классе конфигураций, n_i — число допустимых значений для параметра p_i.
2. Выполняется модификация базового набора, нацеленная на выполнение условий использования параметров.
3. Набор, являющийся числовой матрицей, преобразуется в набор конфигураций при помощи замены номеров значений параметров на сами эти значения. Строка набора при этом соответствует одной конфигурации.

Последний этап не содержит заметных сложностей, поэтому далее разобраны первые два.

3.1. Построение базового набора

На практике используются небольшие значения t = 2-4. Параметры конфигурации упорядочиваются по убыванию количества возможных значений.

Поскольку большинство параметров конфигурации операционной системы часто являются булевскими (они являются флагами, включаемыми/выключаемыми с помощью макроопределений #define/#undef),

для значения t = 2 имеет смысл выделить группу параметров с двумя возможными значениями и использовать для построения части набора, состоящей только из значений этих параметров, наиболее эффективный «булевский» алгоритм [2-4] — он позволяет быстро построить минимальный покрывающий набор глубины 2 для произвольного количества параметров, принимающих только два значения. Для t > 2 такого эффективного алгоритма не известно, используется «жадный» алгоритм [2,5].

Обычно в практических примерах есть небольшое число (2-5) параметров конфигурации с относительно большим количеством значений (10-30). Для построения части набора, связанной с этими параметрами, используется простая эвристика, основанная на частном случае «аффинного» алгоритма [2,4]. Этот алгоритм позволяет быстро построить покрывающий набор произвольной глубины для p+1 параметра, каждый из которых принимает p значений (где p — степень простого числа). Используемая эвристика построена следующим образом: для любого параметра, кроме первого и имеющих два значения, пусть число значений данного параметра равно n, пока n, деленное на наибольший общий делитель n и (максимального) числа значений первого параметра, не превосходит номера строки в наборе, для выбора значения данного параметра в данной строке используется та же формула, что и в «аффинном» алгоритме.

Для заполнения значений в строках, не подходящих под условие эвристики, а также для дополнения части набора для параметров с двумя значениями (или полного построения этой части для t > 2), используется «жадный» алгоритм [2,5] (поскольку параметры конфигурации упорядочиваются по убыванию количества значений, это вариант алгоритма IPO) в простейшей реализации, без вероятностного подбора кандидатов, — т.е., из множества значений-кандидатов в данную позицию набора выбирается такое значение, которое покрывает наибольшее число еще не покрытых комбинаций значений размера t (если таких значений несколько, берется первое найденное из них). Если же все комбинации t значений с участием данного параметра уже покрыты, оставляется неопределенное значение.

Позиции с неопределенными значениями используются далее для более компактного пополнения набора при учете условий использования.

Как видно, данная техника построена с учетом специфики практических примеров классов конфигураций ОС (большинство параметров имеет два значения, максимальное число значений параметра не превосходит 30, количество параметров с достаточно большим числом значений обычно меньше этого числа) и вряд ли будет давать хорошие результаты (достаточно компактные покрывающие наборы) в более общей ситуации. За счет отсутствия вероятностного перебора вариантов в реализации «жадного» алгоритма, данный метод работает несколько быстрее обычных таких реализаций, хотя получаемые с его помощью наборы могут иметь б*о*льший размер.

3.2. Выполнение условий использования параметров

На втором шаге нужно обеспечить выполнение условий использования параметров. Значение параметра в строке, где не выполняется условие его использования, не играет никакой роли, и поэтому не должно учитываться ни в каких комбинациях.

Для каждой пары (t = 2), тройки (t = 3) или четверки (t = 4) параметров выполняется четыре следующих шага.

- Определение противоречивости условий использования для данного набора параметров. Если эти условия противоречивы, добиваться покрытия различных комбинаций значений этих параметров не нужно — эти комбинации недостижимы.
- Определение ограничений на возможные комбинации в допустимых конфигурациях. Даже если условия использования параметров не противоречат друг другу, они могут ограничить достижимые комбинации их значений, например, если один параметр имеет 5 возможных значений, но в условие его использования входит требование того, что второй параметр имеет фиксированное значение, то достижимых пар их значений всего 5, пытаться покрыть другие пары не имеет смысла.
- Определение того, какие комбинации значений уже покрыты в наборе, полученном на предыдущем этапе, с выполненными условиями использования.
- Пополнение набора отсутствующими в нем достижимыми комбинациями с выполненными условиями использования.

Определение противоречивости условий использования само по себе может оказаться довольно трудной задачей — в общем случае она NP-полна [6], известные алгоритмы ее решения экспоненциальны (см, например, [7-9]).

Однако в примерах классов конфигураций, используемых на практике, условия использования часто оказываются не слишком сложны, в частности, будучи приведенными в конъюнктивную нормальную форму, включают конъюнкты с не более чем одной дизъюнкцией (именно так и оказывается во всех практических примерах). В этом случае можно использовать более эффективные алгоритмы ее решения [10-12], лучшие из которых линейны по размеру анализируемого выражения [12].

Алгоритм [12] очень нагляден:

- каждая дизъюнкция с двумя членами $x \vee y$ преобразуется в две импликации $\neg x \to y$ и $\neg y \to x$;
- строится граф с вершинами, соответствующими литералам (булевским выражениям, не сконструированным при помощи логических операций) или их отрицаниям, и с ребрами, соответствующими импликациям;
- с помощью поиска в глубину находятся компоненты сильной связности в построенном графе (количество операций при этом линейно от числа ребер в графе);
- исходное выражение противоречиво тогда и только тогда, когда есть компонент сильной связности, содержащий одновременно литерал и его отрицание.

В данной работе условия использования параметров приводятся в конъюнктивную нормальную форму (КНФ), набор условий использования группы параметров при этом естественным образом оказывается в такой же форме, после чего для определения его противоречивости применяется алгоритм [12], слегка модифицированный, чтобы учесть конъюнкты, являющиеся литералами или их отрицаниями. При этом структура данных, представляющая анализируемые условия, является объединением графа дизъюнкций и обычного множества литералов и их отрицаний, индексированным по параметрам, задействованным в литералах. После работы алгоритма, если условия оказываются непротиворечивыми, из этой же структуры легко извлекаются ограничения, налагаемые на возможные комбинации значений параметров, входящих в группу.

Дальнейшие шаги вполне тривиальны. При пополнении набора отсутствующими в нем достижимыми комбинациями с выполненными условиями использования активно используются оставленные на предыдущих этапах неопределенные значения — их можно заменить на любое допустимое значение параметра, не сокращая множества покрытых комбинаций. Если добавить новую комбинацию (вместе с нужными для выполнения ее условий использования значениями других параметров) только за счет исправления неопределенных значений не получается, в набор добавляется новая строка, в которой определены значения лишь параметров, входящих в добавляемую комбинацию и условия ее использования, все остальные параметры остаются с неопределенными значениями.

4. Практические результаты

Описанный выше метод построения тестового набора конфигураций был реализован в программном инструменте на языке Java и использован для генерации наборов тестовых конфигураций для операционной системы реального времени на разных аппаратных платформах.

Входными данными для него служат Tcl-скрипты, представляющие собой приложение с графическим интерфейсом для настройки значений параметров. Код этого приложения содержит информацию о всех параметрах конфигурации ОС, их возможных значениях и условиях их использования.

Результатом работы является набор директорий, каждая из которых соответствует одной конфигурации и содержит заголовочные файлы языка C с задаваемым этой конфигурацией набором макроопределений.

В таблице 1 показаны результаты использования этого инструмента для генерации тестовых конфигураций, покрывающих все возможные пары значений параметров (t = 2), в разных классах, соответствующих различным аппаратным платформам.

Столбцы в таблице 1 представляют следующую информацию.

- Столбец Platform — идентификатор платформы.
- Столбец NP — число параметров конфигурации.
- Столбец Params — характеристика параметров конфигурации: по убыванию располагаются количества значений параметров, верхний индекс показывает число параметров с данным количеством значений.
- Столбец C — общее количество условий использования. Видно, что большинство параметров имеет нетривиальные условия использования. Во всех случаях подавляющее большинство условий в КНФ не использует дизъюнкций (для платформ 1 и 2 общее число дизъюнкций во всех условиях — 12, для остальных — 13). На всех платформах есть один параметр, условие использования которого включает 7 простых равенств и 4 дизъюнкции из двух равенств, самые сложные условия без дизъюнкций включают 7 равенств для платформ 1-11, и 6 равенств — для остальных.
- Столбец N1 — число конфигураций в получаемом наборе без учета условий использования.
- Столбец N2 — число конфигураций в получаемом наборе с учетом условий использования.
- Столбец T1 — время генерации набора без учета условий использования в секундах (показано среднее значение за 50 запусков).
- Столбец T2 — время трансформации набора, после которой условия использования выполнены, в секундах (показано среднее значение за 50 запусков).

№	Platform	NP	Params	C	N1	N2	T1	T2
1	i386/i386	228	$4^2 3^6 2^{220}$	172	25	92	0.01	0.31
2	i386/x86	215	$4^2 3^6 2^{207}$	170	23	72	0.01	0.26
3	mips/bt205	373	$17^2 15^1 8^8 7^2 6^1 5^3 4^{19} 3^{21} 2^{316}$	279	322	670	1.57	5.07
4	mips/ bt23-202	331	$17^2 15^1 13^4 9^4 7^2 5^3 4^7 3^{10} 2^{298}$	267	319	653	1.07	4.58
5	mips64/ bt128	548	$17^2 15^1 5^3 4^1 3^{15} 2^{526}$	500	289	556	0.57	7.88
6	mips64/ bt211	334	$17^2 15^1 13^4 9^4 7^2 5^3 4^5 3^{10} 2^{302}$	277	319	726	0.78	4.08
7	mips64/ bt206	334	$17^2 15^1 7^2 6^1 5^3 4^5 3^{18} 2^{302}$	278	289	651	0.58	2.99
8	mips64 /mpon	301	$17^2 15^1 5^3 4^1 3^{15} 2^{279}$	251	289	574	0.33	2.15
9	mips64/ vmips	241	$5^1 4^1 3^6 2^{233}$	203	27	89	0.01	0.42
10	mips64/ cprio64	563	$17^2 15^1 5^3 4^1 3^{15} 2^{541}$	511	289	564	0.58	6.70
11	komdiv64/ bt128	548	$17^2 15^1 5^3 4^1 3^{15} 2^{526}$	500	289	556	0.56	7.80
12	R4000/bt128	419	$17^2 15^1 5^3 4^2 3^{14} 2^{397}$	377	289	541	0.44	7.00
13	R4000/bt206	220	$17^2 15^1 14^{14} 7^2 6^1 5^3 4^6 3^{17} 2^{174}$	155	823	826	1.28	1.65
14	R4000/mpon	183	$17^2 15^1 5^3 4^2 3^{14} 2^{161}$	138	289	543	0.23	0.56
15	R4000/ vmips	113	$5^1 4^2 3^5 2^{105}$	80	28	62	0.01	0.07
16	R4000/ cprio64	434	$17^2 15^1 5^3 4^2 3^{14} 2^{412}$	388	289	549	0.46	5.59

Таблица 1. Результаты применения описанного метода — характеристики использованных классов конфигураций, полученных тестовых наборов и время генерации.

Таблица 1 демонстрирует, что предложенный метод генерации конфигурационных тестов работает достаточно эффективно на реальных примерах. Например, (строка 10) тестовый набор для класса конфигураций, имеющего 563 параметра, 511 из которых имеют нетривиальные условия использования, генерируется за 7 секунд и содержит всего лишь 564 тестовых конфигурации.

5. Заключение

В работе представлен метод автоматической генерации тестов для конфигурационного тестирования на основе покрывающих наборов. Новым элементом предложенного метода является учет условий использования отдельных параметров. Генерация осуществляется в два этапа: создание

покрывающего набора с нужными характеристиками без учета условий использования, основанное на широко известных «булевском», «аффинном» и «жадном» алгоритмах построения покрывающих наборов [2] и оставляющее часть значений неопределенными, и обеспечение выполнения условий использования, использующее доопределение неопределенных значений и пополнение набора.

Использованные при построении покрывающих наборов эвристики хорошо работают при выполнении тех ограничений, которые характерны для области применения метода: подавляющее большинство параметров имеют только два значения, максимальное количество значений параметра не превосходит нескольких десятков, число параметров, для которых возможно более двух значений, не больше этого максимального количества значений. В более общей ситуации эти эвристики не так эффективны.

Для выявления противоречивости условий использования применен алгоритм Аспвала-Пласса-Тарьяна [12], работающий для КНФ с двухчленными дизъюнкциями и имеющий линейную от размера условий сложность.

Предложенный подход не применим для классов конфигураций, имеющих общие ограничения на возможные наборы значений параметров. Расширение его на этот случай потребует, по-видимому, серьезных модификаций. В частности, при большом количестве таких ограничений становится неэффективно строить на первом этапе покрывающий набор с исходными характеристиками, в итоге его потребуется слишком сильно изменить. В таком случае, возможно, будет полезна предварительная трансформация набора параметров, в ходе которой должны быть удалены параметры, полностью вычисляемые (или «почти» вычисляемые, в некотором смысле) на основе других параметров.

Литература

[1] Cohen D. M., Dalal S. R., Parelius J., Patton G. C. The Combinatorial Design Approach to Automatic Test Generation. IEEE Software, 13(5):83-87, 1996.

[2] Кулямин В. В., Петухов А. А. Обзор методов построения покрывающих наборов. Программрование, 37(3):3-41, 2011.

[3] Greene C. Sperner families and partitions of a partially ordered set. In Combinatorics, Hall M. Jr. van Lint J., eds. Dordrecht, Holland, 1975, pp. 277-290.

[4] Hartman A. Software and Hardware Testing Using Combinatorial Covering Suites. Proc. of Graph Theory, Combinatorics and Algorithms: Interdisciplinary Applications, 2005, pp. 266-327.

[5] Bryce R. C., Colbourn C. J., Cohen M. B. A Framework of Greedy Methods for Constructing Interaction Test Suites. Proc. of Intl. Conf. on Software Engineering (ICSE'05), 2005, pp. 146-155.

[6] Cook S. The complexity of theorem proving procedures. Proc of 3-rd Annual ACM Symposium on Theory of Computing, 1971. pp. 151-158.

[7] Davis M., Logemann G., Loveland D. A machine program for theorem-proving. Communications of the ACM 5(7):394-397, 1962.

[8] Schoning T. A probabilistic algorithm for k-SAT and constraint satisfaction problems. Proc. of 40-th Annual Symposium on Foundations of Computer Science, 1999, pp. 410-414.

[9] Paturi R., Pudlak P., Saks M. E., Zani F. An improved exponential-time algorithm for k-SAT. Journal of the ACM, 52(3):337-364, 2005.

[10] Krom M. R. The Decision Problem for a Class of First-Order Formulas in Which all Disjunctions are Binary. Zeitschrift für Mathematische Logik und Grundlagen der Mathematik, 13(1-2):15-20, 1967.

[11] Even S., Itai A., Shamir A. On the complexity of time table and multi-commodity flow problems. SIAM Journal on Computing, 5(4):691-703, 1976.

[12] Aspvall B., Plass M. F., Tarjan R. E. A linear-time algorithm for testing the truth of certain quantified boolean formulas. Information Processing Letters, 8(3):121-123, 1979.

Combinatorial generation of operation system software configurations

V.V. Kuliamin
ISP RAS, Moscow, Russia

Abstract. The paper presents an operating system configuration test generation method based on construction of covering arrays, that is ensuring coverage of all pairs, triple, etc. of configuration parameters values. The method combines known optimal algorithm for binary pairwise coverage array generation with further greedy construction of non-binary part of the array. Novelty of the method proposed is taking into account parameters usage conditions, that is constraints on parameters values, which should be obeyed to force the operating system under test to use the value of certain parameter. Usage conditions require to change both accounting of tuples covered and test construction process, which includes now the parametres assignments making all the necessary constraints to hold. Construction of such an assignment uses satisfiability check based on well-known Aspvall-Plass-Tarjan algorithm. The method proposed is applied in configuration test generation for real-time operating system with several hundreds configuration parameters, the results of this application demonstrate effectiveness of the method — usually several seconds is enough for generation of up to thousand different configurations (taking in account only analysis and generation, without input and output phases) where dozens of usage conditions are satisfied.

Keywords: configuration testing; covering array; test generation

References

[1]. Cohen D. M., Dalal S. R., Parelius J., Patton G. C. The Combinatorial Design Approach to Automatic Test Generation. IEEE Software, 13(5):83-87, 1996.
[2]. Kuliamin V., Petukhov A. A survey of methods for constructing covering arrays. Programming and Computer Software 37(3): 121-146, 2011.
[3]. Greene C. Sperner families and partitions of a partially ordered set. In Combinatorics, Hall M. Jr. van Lint J., eds. Dordrecht, Holland, 1975, pp. 277-290.
[4]. Hartman A. Software and Hardware Testing Using Combinatorial Covering Suites. Proc. of Graph Theory, Combinatorics and Algorithms: Interdisciplinary Applications, 2005, pp. 266-327.
[5]. Bryce R. C., Colbourn C. J., Cohen M. B. A Framework of Greedy Methods for Constructing Interaction Test Suites. Proc. of Intl. Conf. on Software Engineering (ICSE'05), 2005, pp. 146-155.
[6]. Cook S. The complexity of theorem proving procedures. Proc of 3-rd Annual ACM Symposium on Theory of Computing, 1971. pp. 151-158.
[7]. Davis M., Logemann G., Loveland D. A machine program for theorem-proving. Communications of the ACM 5(7):394-397, 1962.

[8]. Schoning T. A probabilistic algorithm for k-SAT and constraint satisfaction problems. Proc. of 40-th Annual Symposium on Foundations of Computer Science, 1999, pp. 410-414.
[9]. Paturi R., Pudlak P., Saks M. E., Zani F. An improved exponential-time algorithm for k-SAT. Journal of the ACM, 52(3):337-364, 2005.
[10]. Krom M. R. The Decision Problem for a Class of First-Order Formulas in Which all Disjunctions are Binary. Zeitschrift für Mathematische Logik und Grundlagen der Mathematik, 13(1-2):15-20, 1967.
[11]. Even S., Itai A., Shamir A. On the complexity of time table and multi-commodity flow problems. SIAM Journal on Computing, 5(4):691-703, 1976.
[12]. Aspvall B., Plass M. F., Tarjan R. E. A linear-time algorithm for testing the truth of certain quantified boolean formulas. Information Processing Letters, 8(3):121-123, 1979.