Верификация и анализ вариабельных операционных систем[1]

[1,2,3] *В.В. Кулямин <kuliamin@ispras.ru>*
[1,4] *Е.М. Лаврищева <lavr@ispras.ru>*
[1] *В.С. Мутилин <mutilin@ispras.ru>*
[1,2,3] *А.К. Петренко <petrenko@ispras.ru>*

[1] *Институт системного программирования РАН,*
109004, Россия, г. Москва, ул. А. Солженицына, д. 25
[2] *Московский государственный университет имени М.В. Ломоносова,*
119991, Россия, Москва, Ленинские горы, д. 1
[3] *НИУ Высшая школа экономики,*
101000, Россия, Москва, ул. Мясницкая, д. 20
[4] *Московский физико-технический институт (гос. университет),*
141700, Россия, Московская обл., г. Долгопрудный, Институтский пер., д. 9

Аннотация. В данной работе рассматриваются проблемы верификации и анализа сложных операционных систем с учетом их вариабельности, или наличия большого количества разнообразных конфигураций. Исследуются методы, позволяющие преодолеть эти проблемы, проводится их обзор и классификация. Выделены классы методов, использующих для анализа инструменты, не учитывающие вариабельность, и выборки вариантов системы и методов, использующих специализированные инструменты, учитывающие вариабельность. Как наиболее перспективные с точки зрения масштабируемости, выделены техники анализа, использующие выборки вариантов системы, обеспечивающие покрытие ее кода и комбинаций значений конфигурационных параметров, а также специализированные, учитывающие вариабельность кода техники анализа с итеративным уточнением модели поведения системы на основе контрпримеров.

Ключевые слова: операционная система; семейство систем; модель вариабельности; верификация; статический анализ; проверка моделей; проверка типов; покрытие кода; покрывающий набор; итеративное уточнение модели на основе контрпримеров.

DOI: 10.15514/ISPRAS-2016-28(3)-12

[1] Работа поддержана грантом Российского фонда фундаментальных исследований № 16-01-00352.

Для цитирования: Кулямин В.В., Лаврищева Е.М., Мутилин В.С., Петренко А.К. Верификация и анализ вариабельных операционных систем. Труды ИСП РАН, том 28, вып. 3, 2016 г., стр. 189-208. DOI: 10.15514/ISPRAS-2016-28(3)-12

1. Введение

В современном мире программное обеспечение (ПО) используется для решения все более ответственных и сложных задач, что обуславливает постоянный рост сложности самих программ. При этом процессы разработки, анализа и сопровождения ПО также усложняются, и требуются специальные меры для снижения темпов роста их стоимости и трудозатрат. Одним из методов снижения затрат на создание и сопровождение сложных систем, решающих большое количество разнообразных задач, является создание *вариабельных систем* (или *семейств систем*, software product families, software product lines) [1-4]. Экономия здесь достигается на том, что разработчики пытаются сразу создавать многократно используемые элементы нескольких систем с близким набором функций, предназначенных для различных групп пользователей, сокращая таким образом затраты на создание всех этих систем в совокупности. На сегодняшний день предложены многочисленные техники разработки вариабельных систем различных типов, включая операционные системы (ОС).

При разработке вариабельных систем важнейшую роль играют *модель вариабельности* (variability model) и *механизм обеспечения вариабельности* (variation mechanism). Модель вариабельности задает пространство возможных вариантов данного семейства систем. Обычно она определяется набором характеристик (features) или конфигурационных параметров, множествами их возможных значений и ограничениями на возможные комбинации этих значений, каждый вариант системы при этом соответствует некоторому набору значений всех характеристик. Механизм вариабельности обеспечивает возможность построения всех возможных вариантов системы из ограниченного набора создаваемых и сопровождаемых артефактов.

Механизмы обеспечения вариабельности достаточно разнородны, но могут быть разделены на три группы [5]: *интеграционные*, основанные на отдельной разработке элементов для различных вариантов и дальнейшей их интеграции в разных комбинациях, *генеративные*, использующие параметризацию для генерации различных вариантов одного элемента из общей основы, и *смешанные*, определенным образом объединяющие обе эти техники. В области операционных систем (здесь и далее под операционной системой мы понимаем ядро и базовые библиотеки ОС, предоставляющие приложениям интерфейсы для работы с вычислительными ресурсами и аппаратным обеспечением) в качестве механизма вариабельности смешанного типа широко используется механизм условной компиляции языков C/C++ (на основе макросов #ifdef, #elif, #else). Он позволяет на этапе сборки составлять код, объединяющий различные элементы, задаваемые набором значений

характеристик, являющихся в этом случае параметрами условной компиляции (определяемыми с помощью макросов #define и #undef, а также параметров запуска препроцессора).

Сложность моделей вариабельности современных операционных систем очень высока, например, ядро Linux версии 2.6.32 имеет 6319 характеристик, более 10000 ограничений, которые могут задействовать до 22-х отдельных характеристик, при этом большинство характеристик зависит как минимум от 4-х других, а максимальная глубина дерева зависимостей равна 8 [6]. Такая сложность приводит к большому количеству ошибок, связанных, прежде всего, с трудностями учета всех факторов, которые должен принимать во внимание разработчик отдельного элемента кода. Соответственно, для выявления и преодоления этих ошибок необходимо использовать специализированные техники анализа и верификации. Сложности анализа, характерные для систем с таким механизмом вариабельности, возникают из-за огромного размера пространства допустимых вариантов (что делает совершенно нереалистичным проверку их всех) и одновременной невозможности проверки отдельных фрагментов, из которых собирается код системы. Поскольку используется условная компиляция, каждый фрагмент не обязан являться отдельным компонентом с определенным поведением, которое можно было бы проанализировать отдельно от остального кода, обычно такие фрагменты являются лишь вставками в общий код, и могут быть проверены лишь в определенных комбинациях друг с другом. Необходимость решения этих проблем накладывает на инструменты и методы анализа, применяемые для сложных вариабельных операционных систем и системного ПО вообще, особые требования. Эти требования специфичны именно для анализа и верификации — методы, применяемые для создания таких систем, сами по себе не облегчают их анализ [7,8]. Основная цель данного исследования — определение и развитие масштабируемых методов анализа вариабельных операционных систем, позволяющих справиться с описанными проблемами.

Далее мы кратко рассматриваем основные элементы моделей вариабельности и используемые в области системного ПО языки для их описания, проводим обзор методов анализа и верификации сложного вариабельного системного ПО и выделяем из описанных в литературе методов наиболее перспективные для дальнейшего развития.

2. Модели вариабельности и языки их описания

Для продуктивной работы по созданию или сопровождению определенной вариабельной системы (семейства систем) необходимо понимание ее модели вариабельности, описывающей все многообразие вариантов системы (систем, входящих в данное семейство). Модели вариабельности операционных систем по существу не отличаются от моделей вариабельности, используемых для систем других типов. Обычно в рамках такой модели один вариант системы

соответствует некоторому набору значений определенных характеристик или конфигурационных параметров, поэтому модель вариабельности также часто называется моделью характеристик [9] или моделью конфигураций. Сама модель при этом описывается множеством используемых характеристик, множествами возможных значений для каждой характеристики и набором ограничений на допустимые комбинации значений характеристик. Часть характеристик соответствует видимым пользователю функциям системы, а другая часть - различным альтернативным способам реализации этих функций, а также решениям, принятым при разработке системы и не имеющим непосредственного влияния на ее восприятие пользователями (только косвенное). Для операционных систем примерами характеристик могут быть: поддержка работы с сетью, поддержка многих потоков, поддержка отладки системных процессов и пр.

Большое количество характеристик обычно имеют булевские значения (характеристика может быть включена или выключена в данном варианте), реже встречаются числовые значения или строки, чаще всего одна характеристика может иметь лишь небольшое конечное множество значений. Довольно часто встречаются ограничения, предписывающие использовать значения одной характеристики только при включенной другой или нескольких других (первая характеристика зависит от второй, первая имеет смысл только при включенной второй, например, возможность удаленного доступа к системному журналу может иметься только при включенной поддержке работы с сетью и включенной поддержке ведения журнала). Чаще всего из нескольких выбирается одна, наиболее значимая зависимость, что позволяет оформлять подобного рода ограничения в виде структуры дерева (иерархии) зависимостей на характеристиках. В этом дереве зависимые характеристики считаются дочерними по отношению к той, от которой они зависят, корнем дерева считается некоторая абстрактная характеристика (соответствующая самому рассматриваемому семейству систем), промежуточными узлами чаще всего становятся только булевские характеристики (которые можно включить/выключить), а характеристики с другими множествами значений могут быть только листьями этого дерева. Изредка небулевские характеристики тоже могут быть промежуточными узлами дерева, но при этом во множестве их значений должно быть выделенное значение, соответствующее «выключению» этой характеристики, остальные значения при этом означают разные варианты ее «включения». Довольно часто встречаются также ограничения вида «включение/исключение»: включение одной характеристики должно всегда сопровождаться включением другой и/или выключением нескольких других характеристик.

Известно достаточно много языков для описания моделей характеристик [10,11], предоставляющих наглядные средства для описания указанных выше широко встречающихся видов ограничений и зависимостей.

В области операционных систем (и системного ПО вообще) для этих целей чаще всего [12] используются языки Kconfig [13,14], применяемый для описания возможных конфигураций ядра Linux с 2002 г., и CDL (Component Definition Language) [15], используемый в рамках открытой операционной системы реального времени eCos [16] для встроенных систем. Оба языка поддерживают все основные элементы метода FODA (Feature-Oriented Domain Analysis) [9], описанные выше. Однако поддержка сложных ограничений, связывающих характеристики, не являющиеся детьми одного родителя в дереве зависимостей, в инструментах, работающих с Kconfig, несколько хуже, что часто приводит к ошибкам при описании и обработке сложных конфигураций [6]. Детальное сопоставление возможностей обоих языков и обзор их использования на практике приведены в [12], на Рис. 1 показан пример описания небольшой модели на обоих языках, взятый из [12].

```
k-1  menuconfig MISC_FILESYSTEMS
k-2    bool "Miscellaneous filesystems"
k-3
k-4  if MISC_FILESYSTEMS
k-5
k-6    config JFFS2_FS
k-7     tristate "Journalling Flash File System" if MTD
k-8     select CRC32 if MTD
k-9
k-10
k-11
k-12
k-13   config JFFS2_FS_DEBUG
k-14    int "JFFS2 Debug level (0=quiet, 2=noisy)"
k-15    depends on JFFS2_FS
k-16    default 0
k-17    range 0 2
k-18    --- help ---
k-19      Debug verbosity of ...
k-20
k-21
k-22   config JFFS2_FS_WRITEBUFFER
k-23    bool
k-24    depends on JFFS2_FS
k-25    default HAS_IOMEM
k-26
k-27
k-28   config JFFS2_COMPRESS
k-29    bool "Advanced compression options for JFFS2"
k-30    depends on JFFS2_FS
k-31
k-32   config JFFS2_ZLIB
k-33    bool "Compress w/zlib..." if JFFS2_COMPRESS
k-34    depends on JFFS2_FS
k-35    select ZLIB_INFLATE
k-36    default y
k-37
k-38   choice
k-39    prompt "Default compression" if JFFS2_COMPRESS
k-40    default JFFS2_CMODE_PRIORITY
k-41    depends on JFFS2_FS
k-42    config JFFS2_CMODE_NONE
k-43     bool "no compression"
k-44    config JFFS2_CMODE_PRIORITY
k-45     bool "priority"
k-46    config JFFS2_CMODE_SIZE
k-47     bool "size (EXPERIMENTAL)"
k-48   endchoice
k-49 endif
```

```
c-1  cdl_component MISC_FILESYSTEMS {
c-2   display    "Miscellaneous filesystems"
c-3   flavor     none
c-4  }
c-5
c-6  cdl_package CYGPKG_FS_JFFS2 {
c-7    display     "Journalling Flash File System"
c-8    requires    CYGPKG_CRC
c-9    implements  CYGINT_IO_FILEIO
c-10   parent      MISC_FILESYSTEMS
c-11   active_if   MTD
c-12
c-13   cdl_option CYGOPT_FS_JFFS2_DEBUG {
c-14    display          "Debug level"
c-15    flavor           data
c-16    default_value    0
c-17    legal_values     0 to 2
c-18    define           CONFIG_JFFS2_FS_DEBUG
c-19    description      "Debug verbosity of...."
c-20   }
c-21
c-22   cdl_option CYGOPT_FS_JFFS2_NAND {
c-23    flavor           bool
c-24    define           CONFIG_JFFS2_FS_WRITEBUFFER
c-25    calculated       HAS_IOMEM
c-26   }
c-27
c-28   cdl_component CYGOPT_FS_JFFS2_COMPRESS {
c-29    display          "Compress data"
c-30    default_value    1
c-31
c-32      cdl_option CYGOPT_FS_JFFS2_COMPRESS_ZLIB {
c-33       display          "Compress data using zlib"
c-34       requires         CYGPKG_COMPRESS_ZLIB
c-35       default_value    1
c-36      }
c-37
c-38      cdl_option CYGOPT_FS_JFFS2_COMPRESS_CMODE {
c-39       display          "Set the default compression mode"
c-40       flavor           data
c-41       default_value    { "PRIORITY" }
c-42       legal_values     { "NONE" "PRIORITY" "SIZE" }
c-43     }
c-44    }
c-45 }
c-46
c-47
c-48
c-49
```

Рис. 1. Описание одной модели на Kconfig (слева) и CDL (справа).

Fig. 1. Description of a model in Kconfig (left) and CDL (right).

2.1. Методы анализа моделей вариабельности

Языки описания моделей вариабельности предоставляют широкий набор возможностей по заданию структуры характеристик и ограничений на их значения. Ограничения чаще всего описываются в виде иерархии и

зависимостей, представленных как логические выражения. Такие ограничения на практике могут быть довольно сложными, что приводит к разного рода ошибкам в моделях вариабельности, например, к противоречиям и неразрешимым зависимостям, из-за которых некоторые характеристики невозможно включить. Для преодоления этих трудностей предназначен автоматический анализ моделей вариабельности, который помогает выявить противоречия и несогласованности в самой модели или проверить допустимости заданной конкретной конфигурации (т.е., что конфигурация удовлетворяет всем ограничениям модели).

Методы анализа моделей вариабельности являются темой активных исследований. Наиболее развитые средства анализа имеются для языков моделирования вариабельности, разрабатываемых в рамках исследовательских проектов [12,17]. Их можно разделить на четыре основных группы [10].

- Анализ на основе пропозициональной логики.
  В рамках этого подхода ограничения модели характеристик транслируются в представляющие их логические формулы, которые затем анализируются с помощью решателей, инструментов для автоматического доказательства различных видов (на основе SAT, BDD и пр.) или инструментов для работы с формальными языками типа Alloy, B или Z. Такие методы довольно широко распространены, их обзор можно найти в [18].
- Анализ на основе онтологий.
  Этот подход использует трансляцию модели вариабельности в модель онтологии. Например, в [19] производится трансляция в OWL DL (Ontology Web Language Description Logic), разрешимое подмножество языка OWL, обладающее достаточной выразительной мощностью. После трансляции становится возможным использовать автоматизированные инструменты анализа онтологий [20], такие как RACER [21].
- Анализ на основе программирования в ограничениях.
  В этом подходе ограничения модели вариабельности транслируются в описание задачи CSP (Constraint Satisfaction Problem), которая затем анализируется с помощью существующих инструментов программирования с ограничениями (constraint programming). См., например, работы [22,23].
- Анализ на основе проверки моделей.
  Некоторые исследователи используют преобразование ограничений модели характеристик в задачи проверки моделей (model checking), которые затем решаются соответствующими инструментами [24,25].
- Применение специализированных алгоритмов.
  Некоторые исследователи предлагают узко специализированные

алгоритмы для решения конкретных задач анализа моделей характеристик, например, для оценки числа допустимых вариантов. Описание таких подходов можно найти в [10].

В инструментах, работающих с практически важными языками Kconfig и CDL, возможности анализа моделей весьма ограничены. Для Kconfig поиск несогласованностей и недопустимости конфигураций выполняются автоматически только для моделей с ограничениями, не включающими характеристики, не имеющие общего родителя. При использовании более общих ограничений их противоречия и соответствие им конфигураций автоматически не выявляются. Пользователи должны вручную отслеживать соблюдение такого рода ограничений.

В CDL имеется поддержка анализа непротиворечивости конфигурации, основанная на пропозициональной логике. При модификации конфигурации инструмент анализа определяет противоречия и подсказывает пользователю способы их разрешения.

В работах [10,12] показано, что для обоих языков Kconfig и CDL нет поддержки более развитых подходов анализа моделей вариабельности, доступных для исследовательских языков, почти неиспользуемых в промышленных проектах. Это говорит о наличии возможностей для существенного развития применяемых на практике языков описания конфигураций системного ПО.

3. Методы верификации и анализа вариабельных ОС

Как уже было сказано, основные проблемы анализа и верификации сложного вариабельного системного ПО связаны с невозможностью получить значимую информацию, анализируя комбинируемые в рамках вариантов фрагменты кода по отдельности, из-за отсутствия у них свойств, переносимых на поведение самих вариантов ПО, и с неспособностью проверить каждый из вариантов ПО в силу их огромного количества. Решать эти проблемы можно одним из двух способов.

- Использовать без модификаций те же инструменты и методы анализа, что и для анализа согласованного кода (одного варианта системы). При этом подвергнуть анализу можно только отдельные варианты, но не все (всех слишком много), соответственно, возникает задача выбора небольшого представительного подмножества из всего пространства допустимых вариантов. Эта задача аналогична задаче выбора небольшого, но представительного множества ситуаций, которые будут использоваться в тестах, из всего гигантского множества ситуаций, возможных при работе тестируемой системы. Вторая задача обычно решается при помощи выбора так называемого *критерия полноты* тестирования, или *критерия покрытия*. Так же и при решении первой логично сформулировать некоторый *критерий покрытия пространства вариантов*, достижение которого (т.е., проведение анализа для набора вариантов, удовлетворяющего этому критерию) по некоторым причинам можно считать достаточным для выявления всех существенных свойств и ошибок, характерных для всего набора возможных вариантов. Идеи, лежащие в основе выбора такого критерия, могут быть различными, но требования к нему такие же, как и к критерию полноты тестирования. Он должен давать, с одной стороны, возможность выявить на удовлетворяющем ему наборе вариантов все существенные особенности поведения систем из семейства и ошибки, и, с другой, возможность выбрать достаточно маленькое множество вариантов, удовлетворяющих этому критерию, чтобы их полный анализ был практически осуществим в рамках ограничений проекта на трудоемкость и стоимость.

  Такие методы можно назвать *анализом выборки вариантов* (sample-based analysis). В обзоре [26] подобные методы названы нацеленными на продукт (product-based).

- Другой способ – модификация методов и инструментов анализа с целью поддержки ими работы с несколькими вариантами одновременно, т.е., внесение в их работу таких изменений, которые позволяют анализировать свойства сразу нескольких вариантов проверяемой системы, используя для экономии усилий близость большинства вариантов друг к другу. В этом случае по-прежнему нельзя надеяться на возможность проверки сразу всех вариантов системы - их слишком много, чтобы можно было учесть все возможные варианты поведения в рамках одного анализа. Однако, можно выделять для каждого выполнения анализа более крупные куски кода, которые уже могут быть компонентами (или группами компонентов) с четко выделенным интерфейсом и определенным поведением, анализируя которое можно адекватно выявлять свойства системы. Поэтому при этом подходе также возникает потребность в критерии покрытия, но не в пространстве допустимых вариантов, а на множестве компонентов системы с их точками вариации - будем называть его далее *критерием покрытия вариаций*. Этот критерий также должен определенным образом гарантировать выявление всех существенных свойств и ошибок вариантов системы и обеспечить возможность выбора лишь небольшого количества групп, каждая из которых в итоге будет подвергнута анализу. Заметим, что особенности используемых инструментов и поведения различных вариантов системы могут потребовать, чтобы эти группы пересекались как по компонентам, так и по точкам вариации.

  Подобные методы иногда называют *учитывающим вариабельность анализом* (variability-aware analysis) [27]. В [26] эти методы названы нацеленными на семейство (family-based).

Обзор описанных в литературе методов анализа вариабельных операционных систем показывает, что оба указанных подхода используются в исследованиях и на практике, однако более активно развиваются методы первого типа, использующие критерии покрытия пространства вариантов. Техники второго типа возникли относительно недавно и пока не достигли масштабируемости, необходимой для поддержки анализа промышленной ОС, такой как Linux.

В обзоре [26] выделен еще один вид методов анализа семейств систем – анализ, нацеленный на характеристики (feature-based), проводимый таким образом, чтобы выявить свойства всех вариантов, обладающих заданным значением выделенной характеристики, безотносительно остальных. Однако используемый для системного ПО механизм вариабельности (условная компиляция) крайне усложняет проведение подобного анализа при наличии многих характеристик, поскольку специфичные только для заданной характеристики свойства очень трудно выделить на фоне сложного поведения, определяемого большим числом характеристик и их взаимосвязями. Поэтому такая разновидность анализа сложного системного ПО практически не встречается и обсуждаться в данной работе не будет.

Еще один обзор инструментов анализа вариабельных систем можно найти в [28]. Некоторую информацию из этого обзора мы используем в дальнейшем, но он, по большей части, рассматривает инструменты анализа вне той системы понятий, которая требуется нам.

3.1. Методы анализа на основе выборки вариантов

Известные методы анализа выборки вариантов делятся на следующие группы [27].

- Использование одной «наиболее представительной» конфигурации.
  В этом случае пытаются выбрать одного представителя проверяемого семейства систем, обладающего как можно большим количеством характерных свойств или включающего как можно больше кода. Чаще всего для выбора такого варианта необходимо привлечение экспертов. В сообществе разработчиков ядра Linux в этом качестве принято использовать конфигурацию allyesconfig, в которой большинство характеристик включено [29].
  Обычно при достаточно высокой сложности модели вариабельности анализ только одного варианта неспособен дать достаточно полную информацию о возможных свойствах всего семейства. Этому часто мешают многочисленные взаимоисключающие характеристики и более сложные ограничения на возможные комбинации их значений. Поэтому в сложных случаях выбор «наиболее представительной» конфигурации может быть достаточно хорошим началом для построения набора вариантов для анализа, но всегда должен дополняться другими вариантами.

- Использование случайных конфигураций.
  Для анализа может использоваться (псевдо-)случайно построенный набор допустимых конфигураций. Для Linux есть инструмент их построения, randconfig, часто используемый при необходимости получить случайные конфигурации [30]. Иногда построение конфигураций, удовлетворяющих всем ограничениям, нетривиально и требует применения сложных эволюционных алгоритмов [31].
  Случайный выбор набора вариантов не гарантирует представительности их поведения для всего анализируемого семейства или достижения определенного критерия покрытия. Тем не менее, он может использоваться как затравка набора вариантов для дальнейшего пополнения на основе выбранного критерия покрытия.
- Использование критериев покрытия кода.
  Критерии покрытия вариантов, использующие покрытие кода, учитывают, насколько в выбранном наборе вариантов представлены различные фрагменты кода, которые могут быть включены или исключены при построении допустимого варианта системы (также могут учитываться различные возможные комбинации таких фрагментов). Эти критерии отличаются от критериев покрытия кода, используемых при тестировании, — первые нацелены на выбор некоторого набора вариантов, содержащих какие-то комбинации фрагментов кода, а вторые на выполнение определенных частей кода. Наиболее широко используемым критерием такого рода является требование того, чтобы каждый (актуальный) фрагмент кода из репозитория программного семейства входил хотя бы в один из вариантов, отобранных для анализа. Несмотря на то, что это не самый сильный критерий — при его использовании могут быть не выявлены ошибки, проявляющиеся только при определенных комбинациях варьируемых фрагментов, — даже его достижение в сложных случаях сопряжено со значительными затратами. Теоретически задача построения минимального такого набора вариантов сводится к некоторой NP-полной задаче [27], поэтому на практике лучше использовать алгоритмы, строящие неоптимальные наборы [32], но даже они не всегда дают удовлетворительный результат. В работе [33] для выбора набора конфигураций ядра Linux, покрывающих как можно больше кода, использовался специализированный инструмент VAMPYR, который смог получить максимальное покрытие кода 91% для архитектуры mips (для архитектур x86 и arm удалось получить всего 88% и 84%). Использование одной конфигурации allyesconfig дало для этой архитектуры покрытие 55% кода (соответственно, 79% и 60%), что показывает, также, насколько на практике одна, даже «самая представительная» конфигурация может мало затрагивать возможные варианты поведения систем семейства. Для повышения

этих значений могут понадобиться гораздо более сложные подходы, способные, например, обеспечить включение кода, выделенного несколькими директивами #if/#ifdef.

- Использование критериев покрытия комбинаций значений характеристик.
  Один из способов задействовать закрываемый условными директивами код – использовать соответствующую комбинацию значений характеристик. Это соображение, а также более общая идея, что, реализуя различные комбинации значений характеристик, можно добиться проявления всех возможных вариантов поведения, служат обоснованием выбора набора вариантов для анализа на основе образуемого ими покрытия различных допустимых комбинаций значений характеристик из модели вариабельности. Математической основой таких методов служат алгоритмы построения *покрывающих наборов* (covering arrays) [34-36], которые часто используются в конфигурационном тестировании [37,38]. Покрывающий набор глубины *t* определяет матрицу значений, в которой столбцы соответствуют характеристикам (значениями в столбце могут быть только допустимые значения соответствующей характеристики), строки – выбираемым вариантам, и каждая комбинация *t* значений в любых разных *t* столбцах обязательно встречается в одной из строк. Простейшим случаем является покрывающий набор глубины 2 (или попарный, pairwise), позволяющий покрыть в рамках набора вариантов, задаваемого его строками, все сочетания пар значений характеристик [39]. Задача построения минимального покрывающего набора NP-полна, но существуют эффективные алгоритмы построения наборов, лишь немного больших, чем минимальные [35-38]. Для сложных моделей вариабельности большей проблемой является удовлетворение всех налагаемых моделью ограничений, поэтому техники создания покрывающих наборов должны дополняться достаточно эффективными методами построения или выбора удовлетворяющих ограничениям конфигураций [40,41].

Как видно, исходя из нацеленности на анализ как можно более широкого набора вариантов поведения, наибольшие перспективы для дальнейшего развития среди методов анализа на основе выборки вариантов имеют методы, основанные на покрытии кода и комбинаций значений характеристик. Вполне возможно создание гибридных техник, совмещающих использование комбинаторной генерации и отслеживание достигаемого покрытия кода.

3.1. Методы анализа, учитывающие вариабельность

Методы анализа и верификации, учитывающие вариабельность проверяемой системы, пытаются проводить проверку сразу многих (или даже всех

возможных) вариантов одновременно, сокращая суммарные затраты на нее за счет большого объема общего кода во всех вариантах.

Учет вариабельности требует при реализации инструментов анализа использования модифицированных деревьев абстрактного синтаксиса, размеченных условиями использования тех или иных узлов, представляющими собой обычно пропозициональные формулы над равенствами характеристик и их возможных значений [27,42]. Опубликованные методы такого рода обычно относятся к методам проверки типов (type safety checking, well-formedness checking) [43-46] или методам проверки моделей (model checking) [47-51], есть также несколько работ, использующих дедуктивную верификацию (theorem proving) [52] и символический мониторинг [53].

Методы анализа, учитывающие вариабельность, обладают двумя существенными недостатками (по сравнению с основанными на выборке вариантов): их применение нуждается в аккуратном использовании полной информации о модели вариабельности, которая часто представлена не только в коде, а и в конфигурационных файлах, и в настройках инструментов сборки, а также требует значительной доработки и усложнения инструментов анализа, чтобы те могли учитывать все используемые характеристики и ограничения.

Наиболее перспективно с точки зрения масштабируемости из этих методов выглядят техники проверки типов, дающие возможность эффективно проверять простейшие свойства корректности, и техники проверки моделей с их итеративным уточнением, [51] требующие минимального вмешательства человека при анализе большого по объему кода со сложным поведением. Остальные методы становятся чрезмерно сложными при учете реалистичных моделей вариабельности.

4. Заключение

В статье рассмотрены задачи верификации и анализа современных промышленных операционных систем с учетом их вариабельности, или наличия большого числа возможных конфигураций. Основные проблемы, стоящие в этой области, — невозможность в разумные сроки провести анализ всех допустимых вариантов системы и невозможность выявить значимые свойства при анализе отдельных фрагментов кода, из которых собираются эти варианты.

Методы анализа, способные справиться с этими проблемами, делятся на две группы: анализ некоторой выборки вариантов из всех возможных и анализ кода с учетом его вариативности. На основе проведенного обзора таких методов, учитывая большой объем кода и сложность современного системного ПО, а также нацеленность на проведение как можно более полного анализа поведения всех конфигураций системы, мы выбрали несколько методов, наиболее перспективных для дальнейшего развития. Ими являются хорошо масштабируемые техники, использующие выборку

вариантов на базе покрытия кода и/или на базе покрытия различных комбинаций значений конфигурационных параметров, а также техники анализа кода, использующие итеративное уточнение моделей на основе контрпримеров.

Список литературы

[1]. Jacobson I., Griss M., Jonsson P. Software Reuse, Architecture, Process and Organization for Business Success. Addison-Wesley, 1997. ISBN-13: 978-0201924763.

[2]. Bosch J. Design and Use of Software Architectures: Adopting and Evolving a Product Line Approach. Pearson Education, 2000. ISBN-13: 978-0201674941.

[3]. Clements P., Northrop L. Software Product Lines: Practices and Patterns. SEI Series in Software Engineering, Addison-Wesley, 2001. ISBN-13: 978-0201703320.

[4]. Pohl K., Böckle G., van der Linden F. J. Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag, 2005. DOI: 10.1007/3-540-28901-1.

[5]. Bachmann F., Clements P. Variability in software product lines. CMU/SEI Technical Report CMU/SEI-2005-TR-012, 2005.

[6]. Lotufo R., She S., Berger T., Czarnecki K., Wąsowski A. Evolution of the Linux kernel variability model. Proc. of SPLC'10, LNCS 6287:136-150, Springer, 2010. DOI: 10.1007/978-3-642-15579-6_10.

[7]. Лаврищева Е. М., Коваль Г.И., Слабоспицькая О.О., Колесник А.Л. Особенности процессов управления при создании семейств программных систем. Проблемы программирования, (3):40-49, 2009.

[8]. Лаврищева Е.М., Слабоспицькая О.А., Коваль Г.И., Колесник А.А. Теоретические аспекты управления вариабельностью в семействах программных систем. Вестник КНУ, серия физ.–мат. наук, (1):151-158, 2011.

[9]. Kang K., Cohen S., Hess J., Novak W., Peterson S. Feature-oriented domain analysis (FODA) feasibility study. CMU/SEI Technical Report CMU/SEI-90-TR-21, 1990.

[10]. Benavides D., Segura S., Ruiz-Cortés A. Automated analysis of feature models 20 years later: a literature review. Information Systems, 35(6):615–636, 2010. DOI: 10.1016/j.is.2010.01.001.

[11]. Chen L., Babar M.A. A systematic review of evaluation of variability management approaches in software product lines. Information and Software Technology, 53(4):344–362, 2011. DOI: 10.1016/j.infsof.2010.12.006.

[12]. Berger T., She S., Lotufo R., Wąsowski A., Czarnecki K. A study of variability models and languages in the systems software domain. IEEE Transactions on Software Engineering, 39(12):1611-1640, 2013. DOI: 10.1109/TSE.2013.34.

[13]. Zippel R. et al. Kconfig language. https://www.kernel.org/doc/Documentation/kbuild/kconfig-language.txt.

[14]. She S., Berger T. Formal semantics of the Kconfig language. Technical note, University of Waterloo, 2010.

[15]. Veer B., Dallaway J. The eCos component writer's guide, 2000.

[16]. eCos home page. http://ecos.sourceware.org.

[17]. Berger T., Rublack R., Nair D., Atlee J.M., Becker M., Czarnecki K., Wąsowski A. A survey of variability modeling in industrial practice. Proc. of the 7-th Intl. Workshop on Variability Modelling of Software-intensive Systems (VaMoS'2013), article No. 7, ACM 2013. DOI: 10.1145/2430502.2430513.

[18]. Batory D. Feature models, grammars, and propositional formulas. Proc. of the 9-th Intl. Conf. on Software Product Lines (SPLC'05), LNCS 3714, pp. 7-20, 2005. DOI: 10.1007/11554844_3.

[19]. Wang H., Li Y., Sun J., Zhang H., Pan J. A semantic web approach to feature modeling and verification. Proc. of Workshop on Semantic Web Enabled Software Engineering (SWESE'05), p. 44, 2005.

[20]. OWL DL List of reasoners. http://owl.cs.manchester.ac.uk/tools/list-of-reasoners/

[21]. Haarslev V., Hidde K., Möller R., Wessel M. The RacerPro knowledge representation and reasoning system. Semantic Web, 3(3):267-277, 2012.

[22]. Benavides D., Segura S., Trinidad P., Ruiz-Cortés A. Using Java CSP solvers in the automated analyses of feature models. Generative and Transformational Techniques in Software Engineering, LNCS 4143:399-408. Springer, 2006. DOI: 10.1007/11877028_16.

[23]. White J., Doughtery B., Schmidt D., Benavides D. Automated reasoning for multi-step software product-line configuration problems. Proc. of the 13-th Sofware Product Line Conference, pp. 11-20, 2009.

[24]. Zhang W., Mei H., Zhao H. Feature-driven requirement dependency analysis and high-level software design. Requirements Engineering, 11(3):205-220, 2006. DOI: 10.1007/s00766-006-0033-x.

[25]. Hemakumar A. Finding Contradictions in Feature Models. Proc. of 12-th Intl. Conf. on Software Product Lines (SPLC'2008), v. 2, pp. 183-190, 2008.

[26]. Thüm T., Apel S., Kästner C., Kuhlemann M., Schaefer I., Saake G. A classification and survey of analysis strategies for software product lines. ACM Computing Surveys, 47(1), article No. 6, 2014. DOI: 10.1145/2580950.

[27]. Liebig J., von Rhein A., Kästner C., Apel S., Dörre J., Lengauer C. Scalable analysis of variable software. Proceedings of the 2013 9-th Joint Meeting on Foundations of Software Engineering, pp. 81-91. ACM, 2013. DOI: 10.1145/2491411.2491437.

[28]. Meinicke J., Thüm T., Schröter R., Benduhn F., Saake G. An overview on analysis tools for software product lines. Proc. of the 18-th Intl. Software Product Line Conf.: Companion Vol. for Workshops, Demonstrations and Tools – Vol. 2, pp. 94-101. ACM, 2014. DOI: 10.1145/2647908.2655972.

[29]. Dietrich C., Tartler R., Schröder-Preikshat W., Lohmann D. Understanding Linux feature distribution. Proceedings of the 2012 Workshop on Modularity in Systems Software, pp. 15-20. ACM, 2012. DOI: 10.1145/2162024.2162030.

[30]. Melo J., Flesborg E., Brabrand C., Wąsowski A. A quantitative analysis of variability warnings in Linux. Proceedings of the 10-th International Workshop on Variability Modelling of Software-intensive Systems, pp. 3-8. ACM, 2016. DOI: 10.1145/2866614.2866615.

[31]. Sayyad A. S., Ingram J., Menzies T., Ammar H. Scalable product line configuration: a straw to break the camel's back. Proc. of IEEE/ACM 28-th International Conference on Automated Software Engineering (ASE 2013), pp. 465-474. IEEE, 2013. DOI: 10.1109/ASE.2013.6693104.

[32]. Tartler R., Lohmann D., Dietrich C., Egger C., Sincero J. Configuration coverage in the analysis of large-scale system software. SIGOPS Oper. Syst. Rev., 45(3):10-14, 2012. DOI: 10.1145/2039239.2039242.

[33]. Tartler R., Dietrich C., Sincero J., Schröder-Preikschat W., Lohmann D. Static analysis of variability in system software: the 90000 #ifdefs Issue. Proc. of USENIX Annual Technical Conference (USENIX ATC 14), pp. 421-432, 2014.

[34]. Sloane N.J.A. Covering arrays and intersecting codes. Journal of combinatorial designs, 1(1):51-63, 1993. DOI: 10.1002/jcd.3180010106.
[35]. Hartman A., Raskin L. Problems and algorithms for covering arrays. Discrete Mathematics, 284(1):149-156, 2004. DOI: 10.1016/j.disc.2003.11.029.
[36]. Кулямин В.В., Петухов А.А. Обзор методов построения покрывающих наборов. Программирование 37(3):3-41, 2011. DOI: 10.1134/S0361768811030029.
[37]. Cohen M.B., Gibbons P.B., Mugridge W.B., Colbourn C.J. Constructing test suites for interaction testing. Proc. of 25-th Intl. Conf. on Software Engineering, pp. 38-48. IEEE, 2003. DOI: 10.1109/ICSE.2003.1201186.
[38]. Grindal M., Offutt A.J., Andler S.F. Combination testing strategies: a survey. Software Testing, Verification, and Reliability, 15(3):167-199, 2005. DOI: 10.1002/stvr.319.
[39]. Perrouin G., Oster S., Sen S., Klein J., Baudry B., Le Traon Y. Pairwise testing for software product lines: comparison of two approaches. Software Quality Journal, 20(3-4):605-643, 2012. DOI: 10.1007/s11219-011-9160-9.
[40]. Johansen M.F., Haugen Ø, Fleurey F. Properties of realistic feature models make combinatorial testing of product lines feasible. Proc. of Intl. Conf. on Model Driven Engineering Languages and Systems, pp. 638-652. Springer, 2011. DOI: 10.1007/978-3-642-24485-8_47.
[41]. Кулямин В.В. Комбинаторная генерация программных конфигураций ОС. Труды ИСП РАН, 23:359-370, 2012. DOI: 10.15514/ISPRAS-2012-23-20.
[42]. Brabrand C., Ribeiro M., Tolêdo T., Borba P. Intraprocedural dataflow analysis for software product lines. Proc. Intl. Conf. on Aspect-Oriented Software Development (AOSD), pp. 13-24. ACM, 2012. DOI: 10.1007/978-3-642-36964-3_3.
[43]. Thaker S., Batory D., Kitchin D., Cook W. Safe composition of product lines. Proc. of Intl. Conf. on Generative Programming and Component Engineering (GPCE), pp. 95-104. ACM, 2007. DOI: 10.1145/1289971.1289989.
[44]. Heidenreich F. Towards systematic ensuring well-formedness of software product lines. Proc. of Intl. Workshop on Feature-Oriented Software Development (FOSD), pp. 69-74. ACM, 2009. DOI: 10.1145/1629716.1629730.
[45]. Apel S., Kästner C., Größlinger A., Lengauer C. Type safety for feature-oriented product lines. Automated Software Engineering, 17(3):251-300, 2010. DOI: 10.1007/s10515-010-0066-8.
[46]. Kästner C., Apel S., Thüm T., Saake G. Type checking annotation-based product lines. ACM Trans. Software Engineering and Methodology, 21(3):1-39, 2012. DOI: 10.1145/2211616.2211617.
[47]. Gruler A., Leucker M., Scheidemann K. Modeling and model checking software product lines. Proc. of IFIP Intl. Conf. on Formal Methods for Open Object-based Distributed Systems (FMOODS), pp. 113-131. Springer, 2008. DOI: 10.1007/978-3-540-68863-1_8.
[48]. Lauenroth K., Toehning S., Pohl K.. Model checking of domain artifacts in product line engineering. Proc. Intl. Conf. on Automated Software Engineering (ASE), pp. 269-280. IEEE, 2009. DOI: 10.1109/ASE.2009.16.
[49]. Classen A., Heymans P., Schobbens P.-Y., Legay A., Raskin J.-F. Model checking lots of systems: efficient verification of temporal properties in software product lines. Proc. Int. Conf. Software Engineering (ICSE), pp. 335-344. ACM, 2010. DOI: 10.1145/1806799.1806850.
[50]. Apel S., Speidel H., Wendler P., von Rhein A., Beyer D. Detection of feature interactions using feature-aware verification. Proc. of Intl. Conf. on Automated Software Engineering (ASE), pp. 372-375. IEEE, 2011. DOI: 10.1109/ASE.2011.6100075.

[51]. Cordy M., Heymans P., Legay A., Schobbens P.-Y., Dawagne B., Leucker M. Counterexample guided abstraction refinement of product-line behavioural models. Proc. of 22-nd ACM SIGSOFT Intl. Symposium on Foundations of Software Engineering (FSE 2014), pp. 190-201. ACM, 2014. DOI: 10.1145/2635868.2635919.
[52]. Thüm T., Schaefer I., Apel S., Hentschel M. Family-based deductive verification of software product lines. Proc. of the 11-th Intl. Conf. on Generative Programming and Component Engineering (GPCE '12), pp. 11-20. ACM, 2012. DOI: 10.1145/2371401.2371404.
[53]. Kästner C., von Rhein A., Erdweg S., Pusch J., Apel S., Rendel T., Ostermann K. Toward variability-aware testing. Proc. of Intl. Workshop on Feature-Oriented Software Development (FOSD), pp. 1-8. ACM, 2012. DOI: 10.1145/2377816.2377817.

Verification and analysis of variable operating systems

[1,2,3] *V.V. Kuliamin <kuliamin@ispras.ru>*
[1,4] *E.M. Lavrischeva <lavr@ispras.ru>*
[1] *V.S. Mutilin <mutilin@ispras.ru>*
[1,2,3] *A.K. Petrenko <petrenko@ispras.ru>*

[1] *Institute for System Programming RAS,*
A. Solzhenitsyn str., 25, Moscow, 109004, Russia
[2] *Lomonosov Moscow State University,*
Leninskie gory, 1, Moscow, 119991, Russia
[3] *FCS NRU Higher School of Economics,*
Myasnitskaya str., 20, Moscow, 101000, Russia
[4] *Moscow Institute of Physics and Technology,*
Institutskiy per., 9, Dolgoprudny, Moscow reg., 141700, Russia

Abstract. This paper regards problems of analysis and verification of complex modern operating systems, which should take into account variability and configurability of those systems. The main problems of current interest are related with conditional compilation as variability mechanism widely used in system software domain. It makes impossible fruitful analysis of separate pieces of code combined into system variants, because most of these pieces of code has no interface and behavior. From the other side, analysis of all separate variants is also impossible due to their enormous number. The paper provides an overview of analysis methods that are able to cope with the stated problems, distinguishing two classes of such approaches: analysis of variants sampling based on some variants coverage criteria and variation-aware analysis processing many variants simultaneously and using similarities between them to minimize resources required. For future development we choose the most scalable technics, sampling analysis based on code coverage and on coverage of feature combinations and variation-aware analysis using counterexample guided abstraction refinement approach.

Keywords: operating system; software product family; variability model; software verification; static analysis; model checking; type safety checking; source code coverage; covering array; counterexample-guided abstraction refinement.

DOI: 10.15514/ISPRAS-2016-28(3)-12

For citation: Kuliamin V.V., Lavrischeva E.M., Mutilin V.S., Petrenko A.K. [Verification and analysis of variable operating systems]. *Trudy ISP RAN/Proc. ISP RAS*, vol. 28, issue 3, 2016, pp. 189-208 (in Russian). DOI: 10.15514/ISPRAS-2016-1(2)-12

References

[1]. Jacobson I., Griss M., Jonsson P. Software Reuse, Architecture, Process and Organization for Business Success. Addison-Wesley, 1997. ISBN-13: 978-0201924763.

[2]. Bosch J. Design and Use of Software Architectures: Adopting and Evolving a Product Line Approach. Pearson Education, 2000. ISBN-13: 978-0201674941.

[3]. Clements P., Northrop L. Software Product Lines: Practices and Patterns. SEI Series in Software Engineering, Addison-Wesley, 2001. ISBN-13: 978-0201703320.

[4]. Pohl K., Böckle G., van der Linden F. J. Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag, 2005. DOI: 10.1007/3-540-28901-1.

[5]. Bachmann F., Clements P. Variability in software product lines. CMU/SEI Technical Report CMU/SEI-2005-TR-012, 2005.

[6]. Lotufo R., She S., Berger T., Czarnecki K., Wąsowski A. Evolution of the Linux kernel variability model. Proc. of SPLC'10, LNCS 6287:136-150, Springer, 2010. DOI: 10.1007/978-3-642-15579-6_10.

[7]. Lavrischeva E.M., Koval' G.I., Slabospitskaya O.O., Kolesnik A.L. [Product Line Development Management Specifics]. Problemy programmiovaniya [Problems of Software Developement], (3):40-49, 2009 (in Ukrainian).

[8]. Lavrischeva E.M., Slabospitskaya O.O., Koval' G.I., Kolesnik A.L. [Theoretical Aspects of Variability Management in Product Lines]. Vesnik KNU seria fiz.-mat. nauk [Notes of KNU, series on maths and physics], (1):151-158, 2011 (in Ukrainian).

[9]. Kang K., Cohen S., Hess J., Novak W., Peterson S. Feature-oriented domain analysis (FODA) feasibility study. CMU/SEI Technical Report CMU/SEI-90-TR-21, 1990.

[10]. Benavides D., Segura S., Ruiz-Cortés A. Automated analysis of feature models 20 years later: a literature review. Information Systems, 35(6):615–636, 2010. DOI: 10.1016/j.is.2010.01.001.

[11]. Chen L., Babar M.A. A systematic review of evaluation of variability management approaches in software product lines. Information and Software Technology, 53(4):344–362, 2011. DOI: 10.1016/j.infsof.2010.12.006.

[12]. Berger T., She S., Lotufo R., Wąsowski A., Czarnecki K. A study of variability models and languages in the systems software domain. IEEE Transactions on Software Engineering, 39(12):1611-1640, 2013. DOI: 10.1109/TSE.2013.34.

[13]. Zippel R. et al. Kconfig language. https://www.kernel.org/doc/Documentation/kbuild/kconfig-language.txt.

[14]. She S., Berger T. Formal semantics of the Kconfig language. Technical note, University of Waterloo, 2010.

[15]. Veer B., Dallaway J. The eCos component writer's guide, 2000.

[16]. eCos home page. http://ecos.sourceware.org.

[17]. Berger T., Rublack R., Nair D., Atlee J.M., Becker M., Czarnecki K., Wąsowski A. A survey of variability modeling in industrial practice. Proc. of the 7-th Intl. Workshop on Variability Modelling of Software-intensive Systems (VaMoS'2013), article No. 7, ACM 2013. DOI: 10.1145/2430502.2430513.

[18]. Batory D. Feature models, grammars, and propositional formulas. Proc. of the 9-th Intl. Conf. on Software Product Lines (SPLC'05), LNCS 3714, pp. 7-20, 2005. DOI: 10.1007/11554844_3.

[19]. Wang H., Li Y., Sun J., Zhang H., Pan J. A semantic web approach to feature modeling and verification. Proc. of Workshop on Semantic Web Enabled Software Engineering (SWESE'05), p. 44, 2005.

[20]. OWL DL List of reasoners. http://owl.cs.manchester.ac.uk/tools/list-of-reasoners/

[21]. Haarslev V., Hidde K., Möller R., Wessel M. The RacerPro knowledge representation and reasoning system. Semantic Web, 3(3):267-277, 2012.

[22]. Benavides D., Segura S., Trinidad P., Ruiz-Cortés A. Using Java CSP solvers in the automated analyses of feature models. Generative and Transformational Techniques in Software Engineering, LNCS 4143:399-408. Springer, 2006. DOI: 10.1007/11877028_16.

[23]. White J., Doughtery B., Schmidt D., Benavides D. Automated reasoning for multi-step software product-line configuration problems. Proc. of the 13-th Sofware Product Line Conference, pp. 11-20, 2009.

[24]. Zhang W., Mei H., Zhao H. Feature-driven requirement dependency analysis and high-level software design. Requirements Engineering, 11(3):205-220, 2006. DOI: 10.1007/s00766-006-0033-x.

[25]. Hemakumar A. Finding Contradictions in Feature Models. Proc. of 12-th Intl. Conf. on Software Product Lines (SPLC'2008), v. 2, pp. 183-190, 2008.

[26]. Thüm T., Apel S., Kästner C., Kuhlemann M., Schaefer I., Saake G. A classification and survey of analysis strategies for software product lines. ACM Computing Surveys, 47(1):article 6, 2014. DOI: 10.1145/2580950.

[27]. Liebig J., von Rhein A., Kästner C., Apel S., Dörre J., Lengauer C. Scalable analysis of variable software. Proceedings of the 2013 9-th Joint Meeting on Foundations of Software Engineering, pp. 81-91. ACM, 2013. DOI: 10.1145/2491411.2491437.

[28]. Meinicke J., Thüm T., Schröter R., Benduhn F., Saake G. An overview on analysis tools for software product lines. Proc. of the 18-th Intl. Software Product Line Conf.: Companion Vol. for Workshops, Demonstrations and Tools – Vol. 2, pp. 94-101. ACM, 2014. DOI: 10.1145/2647908.2655972.

[29]. Dietrich C., Tartler R., Schröder-Preikshat W., Lohmann D. Understanding Linux feature distribution. Proceedings of the 2012 Workshop on Modularity in Systems Software, pp. 15-20. ACM, 2012. DOI: 10.1145/2162024.2162030.

[30]. Melo J., Flesborg E., Brabrand C., Wąsowski A. A quantitative analysis of variability warnings in Linux. Proceedings of the 10-th International Workshop on Variability Modelling of Software-intensive Systems, pp. 3-8. ACM, 2016. DOI: 10.1145/2866614.2866615.

[31]. Sayyad A. S., Ingram J., Menzies T., Ammar H. Scalable product line configuration: a straw to break the camel's back. Proc. of IEEE/ACM 28-th International Conference on Automated Software Engineering (ASE 2013), pp. 465-474. IEEE, 2013. DOI: 10.1109/ASE.2013.6693104.

[32]. Tartler R., Lohmann D., Dietrich C., Egger C., Sincero J. Configuration coverage in the analysis of large-scale system software. SIGOPS Oper. Syst. Rev., 45(3):10-14, 2012. DOI: 10.1145/2039239.2039242.

[33]. Tartler R., Dietrich C., Sincero J., Schröder-Preikschat W., Lohmann D. Static analysis of variability in system software: the 90000 #ifdefs Issue. Proc. of USENIX Annual Technical Conference (USENIX ATC 14), pp. 421-432, 2014.

[34]. Sloane N.J.A. Covering arrays and intersecting codes. Journal of combinatorial designs, 1(1):51-63, 1993. DOI: 10.1002/jcd.3180010106.

[35]. Hartman A., Raskin L. Problems and algorithms for covering arrays. Discrete Mathematics, 284(1):149-156, 2004. DOI: 10.1016/j.disc.2003.11.029.

[36]. Kuliamin V.V., Petukhov A.A. A survey of methods for constructing covering arrays. Programming and Computer Software, 37(3):121-146, 2011. DOI: 10.1134/S0361768811030029.

[37]. Cohen M.B., Gibbons P.B., Mugridge W.B., Colbourn C.J. Constructing test suites for interaction testing. Proc. of 25-th Intl. Conf. on Software Engineering, pp. 38-48. IEEE, 2003. DOI: 10.1109/ICSE.2003.1201186.

[38]. Grindal M., Offutt A.J., Andler S.F. Combination testing strategies: a survey. Software Testing, Verification, and Reliability, 15(3):167-199, 2005. DOI: 10.1002/stvr.319.

[39]. Perrouin G., Oster S., Sen S., Klein J., Baudry B., Le Traon Y. Pairwise testing for software product lines: comparison of two approaches. Software Quality Journal, 20(3-4):605-643, 2012. DOI: 10.1007/s11219-011-9160-9.

[40]. Johansen M.F., Haugen Ø, Fleurey F. Properties of realistic feature models make combinatorial testing of product lines feasible. Proc. of Intl. Conf. on Model Driven Engineering Languages and Systems, pp. 638-652. Springer, 2011. DOI: 10.1007/978-3-642-24485-8_47.

[41]. Kuliamin V.V. [Combinatoric generation of operating system software configurations]. Trudy ISP RAN/Proc. ISP RAS, 23:359-370, 2012 (in Russian). DOI: 10.15514/ISPRAS-2012-23-20.

[42]. Brabrand C., Ribeiro M., Tolêdo T., Borba P. Intraprocedural dataflow analysis for software product lines. Proc. Intl. Conf. on Aspect-Oriented Software Development (AOSD), pp. 13-24. ACM, 2012. DOI: 10.1007/978-3-642-36964-3_3.

[43]. Thaker S., Batory D., Kitchin D., Cook W. Safe composition of product lines. Proc. of Intl. Conf. on Generative Programming and Component Engineering (GPCE), pp. 95-104. ACM, 2007. DOI: 10.1145/1289971.1289989.

[44]. Heidenreich F. Towards systematic ensuring well-formedness of software product lines. Proc. of Intl. Workshop on Feature-Oriented Software Development (FOSD), pp. 69-74. ACM, 2009. DOI: 10.1145/1629716.1629730.

[45]. Apel S., Kästner C., Größlinger A., Lengauer C. Type safety for feature-oriented product lines. Automated Software Engineering, 17(3):251-300, 2010. DOI: 10.1007/s10515-010-0066-8.

[46]. Kästner C., Apel S., Thüm T., Saake G. Type checking annotation-based product lines. ACM Trans. Software Engineering and Methodology, 21(3):1-39, 2012. DOI: 10.1145/2211616.2211617.

[47]. Gruler A., Leucker M., Scheidemann K. Modeling and model checking software product lines. Proc. of IFIP Intl. Conf. on Formal Methods for Open Object-based Distributed Systems (FMOODS), pp. 113-131. Springer, 2008. DOI: 10.1007/978-3-540-68863-1_8.

[48]. Lauenroth K., Toehning S., Pohl K.. Model checking of domain artifacts in product line engineering. Proc. Intl. Conf. on Automated Software Engineering (ASE), pp. 269-280. IEEE, 2009. DOI: 10.1109/ASE.2009.16.

[49]. Classen A., Heymans P., Schobbens P.-Y., Legay A., Raskin J.-F. Model checking lots of systems: efficient verification of temporal properties in software product lines. Proc.

Int. Conf. Software Engineering (ICSE), pp. 335-344. ACM, 2010. DOI: 10.1145/1806799.1806850.

[50]. Apel S., Speidel H., Wendler P., von Rhein A., Beyer D. Detection of feature interactions using feature-aware verification. Proc. of Intl. Conf. on Automated Software Engineering (ASE), pp. 372-375. IEEE, 2011. DOI: 10.1109/ASE.2011.6100075.

[51]. Cordy M., Heymans P., Legay A., Schobbens P.-Y., Dawagne B., Leucker M. Counterexample guided abstraction refinement of product-line behavioural models. Proc. of 22-nd ACM SIGSOFT Intl. Symposium on Foundations of Software Engineering (FSE 2014), pp. 190-201. ACM, 2014. DOI: 10.1145/2635868.2635919.

[52]. Thüm T., Schaefer I., Apel S., Hentschel M. Family-based deductive verification of software product lines. Proc. of the 11-th Intl. Conf. on Generative Programming and Component Engineering (GPCE '12), pp. 11-20. ACM, 2012. DOI: 10.1145/2371401.2371404.

[53]. Kästner C., von Rhein A., Erdweg S., Pusch J., Apel S., Rendel T., Ostermann K. Toward variability-aware testing. Proc. of Intl. Workshop on Feature-Oriented Software Development (FOSD), pp. 1-8. ACM, 2012. DOI: 10.1145/2377816.2377817.