На правах рукописи

Мордань Виталий Олегович

МЕТОДЫ ВЕРИФИКАЦИИ ПРОГРАММ НА ОСНОВЕ КОМПОЗИЦИИ ЗАДАЧ ДОСТИЖИМОСТИ

05.13.11 – математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

Автореферат
диссертации на соискание ученой степени
кандидата физико-математических наук

Москва – 2017

Работа выполнена в Федеральном государственном бюджетном образовательном учреждении высшего образования Московский государственный университет имени М.В. Ломоносова и Федеральном государственном бюджетном учреждении науки Институт системного программирования Российской академии наук.

Научный руководитель:	**Петренко Александр Константинович**, доктор физико-математических наук, профессор, заведующий отделом технологий программирования Федерального государственного бюджетного учреждения науки Институт системного программирования Российской академии наук
Официальные оппоненты:	**Соколов Валерий Анатольевич**, доктор физико-математических наук, профессор, заведующий кафедрой теоретической информатики Федерального государственного бюджетного образовательного учреждения высшего образования «Ярославский государственный университет им. П.Г. Демидова»
	Волконский Владимир Юрьевич, кандидат технических наук, старший научный сотрудник, начальник отделения «Системы программирования» публичного акционерного общества «ИНЭУМ им. И.С. Брука»
Ведущая организация:	Федеральный исследовательский центр «Информатика и управление» Российской Академии Наук

Защита состоится «25» мая 2017 г. в 16:30 на заседании диссертационного совета Д 002.087.01 при Институте системного программирования РАН по адресу: 109004, Москва, ул. А. Солженицына, д. 25.

С диссертацией можно ознакомиться в библиотеке и на сайте Федерального государственного бюджетного учреждения науки Институт системного программирования Российской академии наук.

Автореферат разослан «__» _____ 2017 г.

Ученый секретарь
диссертационного совета Д 002.087.01,
кандидат физико-математических наук **Зеленов С.В.**

Актуальность темы

В современном мире программное обеспечение играет значительную роль в жизни общества, поэтому проблема его надежности является крайне актуальной. Хорошо известны примеры, в которых ошибки в программном обеспечении становились причинами катастроф. Поскольку используемое на практике программное обеспечение постоянно развивается и усложняется, то возрастают и потребности в его автоматической проверке.

Одним из примеров ответственного программного обеспечения с требованиями повышенной надежности является ядро операционной системы Linux. Помимо персональных компьютеров операционная система Linux широко распространена на серверах и суперкомпьютерах, а в последнее время и на мобильных устройствах. На данный момент ядро Linux состоит из более 20 миллионов строк кода на языке программирования C. При этом в процесс его разработки вовлечены тысячи разработчиков, каждые 2-3 месяца выходит новая версия, содержащая тысячи изменений. В то же самое время каждая ошибка в ядре Linux является критической и может привести к отказу всей операционной системы, что делает проверку модулей ядра Linux актуальной задачей.

На практике для автоматической проверки программного обеспечения, как правило, применяется тестирование или другие методы поиска ошибок (например, статический анализ кода программ). Однако применение данных подходов не гарантирует отсутствия ошибок, следствием чего является множество примеров, в которых критические ошибки не были выявлены.

Статическая верификация программного обеспечения является средством проверки исходного кода без его выполнения, при этом рассматриваются все возможные пути выполнения программы. Наиболее значимые научные результаты в области статической верификации достигнуты исследователями, развивающими

технологии software model checking (наиболее известные проекты BLAST, CPAchecker, CBMC, SMACK, UAutomizer и др.).

Главное достоинство статической верификации заключается в том, что она нацелена на доказательство корректности программного обеспечения, а не только на поиск часто встречающихся ошибок. Однако в общем случае задача статической верификации не является разрешимой. Помимо этого, на практике статическая верификация требует больших затрат вычислительных ресурсов.

Каждая проверяемая программа должна удовлетворять большому числу специфичных функциональных и нефункциональных требований. Примером нарушения специфичного требования является некорректное использование интерфейсов сердцевины ядра в модулях ядра операционной системы, что может привести к различным негативным последствиям (например, к освобождению некорректного указателя или взаимным блокировкам). В модулях ядра Linux число специфичных требований измеряется сотнями. Статическая верификация является одним из наиболее перспективных средств для проверки выполнения подобных требований.

Для проверки выполнения специфичного требования в программе с помощью статической верификации необходимо по коду программы поставить задачу достижимости, добавив в код вспомогательные проверки на соответствие требованию. Если необходимо проверить несколько требований, то на практике используется последовательная верификация, то есть для каждого требования создается и решается отдельная задача достижимости. При этом многократно проверяется одна и та же программа, в результате чего выполняются однотипные действия, следовательно, вычислительные ресурсы расходуются нерационально. Возможной альтернативой служит верификация композиции нескольких требований. Еще одной общей проблемой статических верификаторов является их остановка после нахождения первого нарушения требования, что ведет к

увеличению числа запусков и вычислительных ресурсов для выполнения верификации.

Таким образом, можно утверждать, что задача верификации композиции требований и разработка методов ее эффективного решения является актуальной темой исследований и разработок.

Цель и задачи работы

Цель работы – разработка методов статической верификации программного обеспечения для проверки соответствия программ композиции требований.

Для достижения данной цели были поставлены следующие задачи:

- Провести анализ существующих методов статической верификации для определения того, насколько они подходят для решения поставленной цели.
- Разработать новые методы верификации программного обеспечения, предназначенные для проверки композиции требований с учетом того, что каждое требование в программе может нарушаться более одного раза.
- Реализовать предложенные методы.
- Дать оценку области применимости предложенных методов и составить рекомендации по их использованию.

Научная новизна работы

Научной новизной обладают следующие результаты работы:

- Метод статической верификации программного обеспечения для обнаружения всех однотипных нарушений проверяемого требования.
- Метод статической верификации программного обеспечения для проверки выполнения композиции требований (условная многоаспектная верификация).
- Метод статической верификации программного обеспечения, расширяющий возможности представления требований в виде их автоматных

спецификаций.

- Метод статической верификации программного обеспечения на основе декомпозиции автоматной спецификации требований на группы требований для совместной верификации внутри группы.
- Сформулированы и доказаны утверждения и теоремы, являющиеся обоснованием корректности предложенных методов.

Теоретическая и практическая значимость

В данной работе были предложены методы статической верификации программного обеспечения, нацеленные на проверку выполнения композиции требований с возможностью нахождения нескольких нарушений требований. Для предложенных методов были сформулированы и доказаны утверждения и теоремы, обосновывающие их корректность. Эти результаты могут использоваться в исследовательских проектах и обучении в курсах формальных методов разработки и анализа программ.

Предложенные методы были реализованы в качестве расширения системы верификации Linux Driver Verification Tools и статического верификатора CPAchecker. Проведенные эксперименты демонстрируют повышение производительности верификации в 4-5 раз.

Результаты данной работы в первую очередь полезны для разработчиков статических верификаторов. Методы проверки выполнения композиции требований позволяют существенно повысить производительность всего процесса верификации при проверке многих требований. Методы обнаружения всех однотипных нарушений требований позволяют выявлять больше ошибок в программном обеспечении с помощью однократного выполнения статической верификации.

Положения, выносимые на защиту

- Методы статической верификации программного обеспечения, основанные на инструментировании исходного кода и предназначенные для обнаружения всех однотипных нарушений (ОВН) и проверки выполнения композиции требований с помощью условной многоаспектной верификации (УМАВ).
- Методы статической верификации программного обеспечения, с использованием формализации требований в виде автоматных спецификаций (АС) и декомпозиции автоматной спецификации на группы требований для совместной верификации (ДАС).
- Теорема о полноте и корректности предложенных методов для требований, удовлетворяющих ограничениям инструментирования исходного кода программы.

Публикации

По теме диссертации автором опубликовано 5 работ (работы [3-5] опубликованы в изданиях из перечня ВАК, они же индексированы в Web of Science и Scopus). В работе [2] описаны предложенные и реализованные автором методы фильтрации трасс ошибок, являющиеся основой для метода ОВН. В работе [3] представлен метод УМАВ и описана его апробация на практике. Возможность совместного использования методов УМАВ и ОВН обоснована автором в работе [4]. В работе [5] автором была предложена основная идея методов АС и ДАС, а также стратегия разбиения спецификации для метода ДАС, оказавшаяся наиболее эффективной в проведенных экспериментах.

В ходе выполнения работы было получено 2 свидетельства о государственной регистрации программы для ЭВМ.

Личный вклад автора

Все представленные в диссертации результаты получены автором лично.

Апробация результатов работы

Основные положения работы докладывались на следующих конференциях и семинарах:

- международный молодежный научный форум «ЛОМОНОСОВ-2014»;
- семинар Института системного программирования РАН (г. Москва, 2014 г.);
- 8-й весенний коллоквиум молодых исследователей в области программной инженерии (SYRCoSE: Spring Young Researchers Colloquium on Software Engineering, г. Санкт-Петербург, 2014 г.);
- 9-й весенний коллоквиум молодых исследователей в области программной инженерии (SYRCoSE: Spring Young Researchers Colloquium on Software Engineering, г. Самара, 2015 г.);
- 10-я международная Ершовская конференция по информатике PSI-2015 (г. Казань, 2015 г.);
- 5-й международный семинар Linux Driver Verification (г. Москва, 2015 г.);
- 1-й международный семинар, посвященный инструменту CPAchecker (г. Пассау, Германия, 2016 г.).

Структура и объем диссертации

Работа состоит из введения, пяти глав, заключения и списка литературы (77 наименований) и трех приложений. Основной текст диссертации (без приложений и списка литературы) занимает 133 страницы, общий объем – 205 страниц.

Краткое содержание диссертации

Во **введении** обосновывается актуальность темы работы, ставятся ее цели и задачи, раскрывается ее практическая значимость.

В **первой главе** рассматривается подход уточнения абстракции по контрпримерам (англ. counterexample guided abstraction refinement, или CEGAR) в качестве базового подхода статической верификации, основная идея которого

заключается в построении абстрактной модели программы, ее итеративном уточнении и доказательстве недостижимости метки ошибки – нарушения проверяемого требования. В случае достижения метки ошибки подход CEGAR выдает трассу ошибки (то есть последовательность операций в исходном коде программы от точки входа к метке ошибки) и завершает работу, поскольку больше нет смысла пытаться доказывать корректность программы. Данный подход на практике применяется только для последовательной верификации (то есть при подготовке и решении отдельной задачи достижимости для каждого требования), но не используется для верификации композиции требований главным образом из-за того, что для этого необходимо строить более точную абстракцию, а это может привести к проблеме экспоненциального роста числа состояний в абстрактной модели программы.

Подход CEGAR способен решать задачи достижимости, поэтому для проверки выполнения требования в произвольной программе для начала необходимо поставить соответствующую задачу достижимости, добавив дополнительные проверки. Для этого требование должно быть формализовано на некотором языке описания требований (например, с помощью аспектно-ориентированного расширения языка C), после чего становится возможным осуществление процедуры инструментирования исходного кода, в которой соответствующие требованию проверки добавляются в программу. Полученная при этом задача достижимости усложняется по сравнению с исходным кодом программы, но открываются достаточно широкие возможности по формализации требования, сопоставимые с возможностями языка программирования исходной программы. Создание «универсальных» задач достижимости, в которых добавлены все имеющиеся формализованные требования, не используется из-за чрезмерного усложнения задач и несовместимости некоторых требований между собой.

Для поддержания всего процесса верификации используются системы верификации, задача которых – подготовить исходный код (то есть выделить подмножество исходного кода программы вместе с опциями препроцессирования и параметрами сборки), поставить для данного кода и проверяемого требования задачу достижимости, решить ее и предоставить результат для анализа человеку. Анализ результатов главным образом нацелен на определение того, соответствуют ли найденные нарушения требований реальным ошибкам или нет. На практике по разным причинам найденная трасса ошибки может являться следствием ложного сообщения об ошибке (например, из-за неполной поддержки верификатором всех конструкций языка программирования). На данный момент в существующих системах верификации производится последовательная верификация всех требований, то есть последовательно подготавливаются и решаются задачи достижимости для проверки различных требований. Именно в систему верификации и должны встраиваться предлагаемые в данной работе методы, в частности фильтрации трасс ошибок, что потребует изменения всего процесса верификации.

Далее рассмотрено решение проблемы переиспользования знаний верификации, отсутствие чего является основной причиной неэффективности последовательной верификации требований, на примере регрессионной верификации. Основная идея методов переиспользования в регрессионной верификации заключается в том, чтобы запомнить промежуточные знания о верификации и затем использовать их для ускорения верификации новых версий программы. Данные методы позволяют существенно сократить время верификации, однако до сих пор они не использовались для верификации одной программы относительно различных требований главным образом потому, что переиспользование некоторых знаний может привести к замедлению верификации (например, если данные знания приводят к созданию более точной абстракции

программы, чем без их переиспользования).

Выявленные проблемы существующих методов не позволяют эффективно решать задачи статической верификации на практике. По результатам обзора формулируются требования для новых методов верификации программного обеспечения для проверки композиции требований, которые предлагаются в следующих главах.

Во **второй главе** «Методы многоаспектной верификации» описываются методы обнаружения всех однотипных нарушений и условной многоаспектной верификации. Основная задача, на решение которой нацелены данные методы, – повышение производительности верификации при сохранении ее качества. Помимо этого, при продолжении верификации после обнаружения первого нарушения требования возникает задача нахождения большего числа различных нарушений требования в коде программы.

Собственно верификации программных модулей предшествует фаза подготовки задачи достижимости. При проверке композиции требований метод подготовки задачи достижимости имеет свои особенности. Глава начинается с описания метода и анализа его корректности. Под корректностью в данном случае понимается то, что результат верификации (то есть либо доказательство выполнения требования в программе, либо нахождение его нарушения) не изменяется при подготовке задач достижимости относительно композиции требований в сравнении с подготовкой задач достижимости относительно каждого требования в отдельности. Для этого формальная модель требования должна удовлетворять ограничениям инструментирования исходного кода, которые запрещают модификацию исходных путей выполнения программы. Для данного способа подготовки задач достижимости справедливо следующее утверждение:

Утверждение 1. Полнота и корректность верификации требований, удовлетворяющих ограничениям инструментирования исходного кода, не

изменяются при подготовке задач достижимости относительно композиции требований в сравнении с подготовкой задач достижимости относительно каждого требования в отдельности.

Здесь и далее сформулированные утверждения и теоремы верны при условии, что после инструментирования целевой программы сложность и размер модифицированной программы не приводят к превышению предельных требований на ресурсы, необходимых для верификации. Например, трансформированная программа вместе с промежуточными данными, которые создаются во время верификации, может потребовать либо слишком много памяти, либо слишком много времени. В реальных условиях при верификации больших программ такие ситуации не являются фатальными, просто отдельные подзадачи верификации, которые превысили выделенные им ресурсы, принудительно завершаются, и вырабатывается вердикт специального вида *Unknown*, то есть, имеется ли в программе искомая ошибка или не имеется – не известно. Такая ситуация также называется «потерей результата». По этой причине в главе 5 и в Приложении Б рассматриваются результаты экспериментов использования предложенных методов при различных ограничениях на выделяемые ресурсы и даются оценки доли потерь результатов верификации в различных условиях.

Метод обнаружения всех однотипных нарушений (ОВН) предназначен для обеспечения возможности продолжения верификации после нахождения нарушения требования и для последующего анализа результата (то есть найденных трасс ошибок). Данный метод требует больше вычислительных ресурсов, т. к. он продолжает работу там, где базовый метод верификации останавливается. Анализ результата необходим для исключения «наведенных» трасс ошибок (то есть соответствующих одному и тому же нарушению требования) и выполняется в два этапа. На первом этапе производится автоматическая фильтрация трасс ошибок (например, с помощью сравнения на

полную эквивалентность) с использованием формального представления трасс. На втором этапе выполняется ручной анализ, нацеленный на выявление причины нарушения требования в трассе, которое нельзя установить автоматически. Таким образом, метод обнаружения всех однотипных нарушений способен находить все нарушения проверяемого требования в программе, однако он требует больше вычислительных ресурсов и проведения ручного анализа результатов.

Для успешного решения задач достижимости, подготовленных относительно композиции требований, необходимо разрешить выявленную во введении проблему экспоненциального роста числа состояний. Для этого была предложена техника многоаспектной верификации, которая в рамках подхода CEGAR разделяет все время верификации на проверку конкретных требований, а следовательно, позволяет ограничить ресурсы на проверку каждого требования в отдельности и тем самым предотвращает экспоненциальный рост числа состояний, вызванный отдельными требованиями. Для этого используется следующее предположение: только одно требование проверяется в каждый момент времени, и алгоритм точно знает только то требование, которое проверялось последним. Последнее проверяемое требование определяется по найденному контрпримеру в цикле CEGAR, который всегда содержит информацию о том, на основании какого именно требования велось построение абстракции. После нахождения нарушения требования или нарушения соответствующего ему ограничения на ресурсы верификация продолжается, но без данного требования, что позволяет получить результат для остальных требований. Для техники многоаспектной верификации была доказана следующая теорема:

Теорема 1. При верификации требований, удовлетворяющих ограничениям инструментирования исходного кода, полнота и корректность техники многоаспектной верификации не изменяются относительно метода последовательной верификации.

Схема доказательства. Пусть дано N требований для верификации в программе P. Для доказательства данной теоремы необходимо показать, что техника многоаспектной верификации не может привести к пропуску ошибок относительно метода последовательной верификации и появлению новых ложных сообщений об ошибках. Из условия для проверяемых требований справедливо утверждение 1, то есть объединение их моделей не влияет на полноту и корректность верификации.

Из доказательства корректности для всех проверяемых требований с помощью техники многоаспектной верификации следует доказательство корректности каждого из проверяемых требований с помощью последовательной верификации. Из нахождения нарушения некоторого требования с помощью техники многоаспектной верификации также следует возможность нахождения точно такого же пути выполнения программы при проверке достижимости только одной метки ошибки (метод последовательной верификации).

Кроме того, стоит заметить, что все добавляемые в технике многоаспектной верификации действия не способны изменять пути выполнения проверяемой программы.

При этом на практике техника многоаспектной верификации не способна предоставить результат для каждого из проверяемых требований (например, проверка только одного требования ведет к исчерпанию доступной памяти или предложенная аппроксимация не справляется с экспоненциальным ростом числа состояний). Для решения данной проблемы и возможности использования предложенной техники на практике был предложен метод условной многоаспектной верификации (УМАВ) на основе идей условной проверки моделей. Данный метод нацелен на запуск техники многоаспектной верификации несколько раз для получения более точного результата – если многоаспектная верификация завершается с ошибкой или не удается предотвратить

экспоненциальный рост числа состояний, то задача достижимости изменяется путем удаления требований, вызвавших некорректное завершение или экспоненциальный рост числа состояний, после чего для нее запускается новая итерация техники многоаспектной верификации. Данная процедура повторяется до тех пор, пока не будет получен результат для каждого проверяемого требования. Основное достоинство данного алгоритма заключается в предоставлении более эффективного решения проблемы экспоненциального роста числа состояний (например, 99% всех состояний в абстракции соответствуют требованию, нарушившему ограничение по времени, поэтому гораздо проще и быстрее полностью перестроить абстракцию для проверки остальных требований). Заметим, что поскольку метод условной многоаспектной верификации состоит в использовании техники многоаспектной верификации несколько раз, то для него также справедлива теорема 1.

Таким образом, методы обнаружения всех однотипных нарушений и условной многоаспектной верификации нацелены на проверку соответствия композиции требований с учетом того, что каждое требование может нарушаться несколько раз. Метод обнаружения всех однотипных нарушений предназначен для нахождения большего числа нарушений требования в программе с помощью однократного проведения статической верификации, но требует больше ресурсов и проведения ручного анализа результатов. Метод условной многоаспектной верификации способен проверять программу относительно композиции требований, удовлетворяющих ограничениям инструментирования исходного кода, с целью повышения производительности верификации. Предложенные методы могут использоваться как по отдельности, так и совместно.

В **третьей главе** «Методы декомпозиции автоматной спецификации» предлагается метод, нацеленный на расширение метода условной многоаспектной верификации за рамки подхода CEGAR, а также на решение проблемы повышения

вычислительной сложности, то есть усложнения задач достижимости из-за инструментирования с использованием многих моделей требований. На практике это приводит к увеличению потребности в необходимых вычислительных ресурсах и тем самым к потере некоторых результатов в случае, когда выделенные ресурсы исчерпываются.

Для решения проблемы усложнения задач достижимости был предложен новый метод формализации требований в виде автоматных спецификаций (АС), основанный на наблюдательных автоматах. Основная идея метода заключается в том, чтобы передавать модель требования верификатору не в форме исходного кода спецификации требования, а в формате внутреннего представления верификатора. Для того чтобы метод мог использоваться для формализации тех же требований, что и в случае инструментирования исходного кода, наблюдательные автоматы были расширены за счет добавления поддержки внутренних переменных и произвольных конструкций языка программирования С. Для метода автоматных спецификаций справедливы следующие утверждения:

Утверждение 2. Для требований, удовлетворяющих ограничениям инструментирования исходного кода, метод автоматных спецификаций (АС) полностью совпадает по возможностям их формализации с методом инструментирования.

Утверждение 3. При верификации требований, удовлетворяющих ограничениям инструментирования исходного кода, полнота и корректность метода автоматных спецификаций (АС) не изменяются относительно метода последовательной верификации на основе инструментирования.

Таким образом, можно утверждать, что метод АС позволяет получить такие же результаты, как и метод условной многоаспектной верификации (УМАВ).

В качестве развития метода автоматных спецификаций был предложен метод декомпозиции автоматной спецификации (ДАС). Он позволяет автоматически

разбить (декомпозировать) спецификации требований на группы требований таким образом, чтобы наилучшим способом укладываться в имеющиеся ресурсы (время и память). При этом часть требований, которые ведут к чрезмерному росту числа состояний при совместной верификации, проверяется по отдельности, а все остальные требования группируются и проверяются совместно. За процесс разбиения требований на группы в данном методе отвечает стратегия разбиения. Для метода декомпозиции автоматной спецификации была доказана следующая теорема:

Теорема 2. При верификации требований, удовлетворяющих ограничениям инструментирования исходного кода, полнота и корректность метода декомпозиции автоматной спецификации (ДАС) не изменяются относительно метода автоматной спецификации (АС).

Схема доказательства. Как и в теореме 1 необходимо показать, что в методе ДАС невозможны появление новых ложных сообщений об ошибках и пропуск ошибок относительно метода АС.

Пусть спецификация состоит из N требований и дана некоторая произвольная стратегия разбиения S, для которой известно, что в каждом новом множестве разбиений максимальное количество требований в одном разбиении уменьшается. Обозначим данное максимальное количество требований в разбиении через R. Докажем данную теорему методом математической индукции.

1. Базис индукции R=1. В данном случае каждое требование помещается в отдельное разбиение, то есть верификация каждого разбиения полностью аналогична методу АС. При этом либо для требования доказывается корректность, либо находится его нарушение, либо исчерпываются выделенные ресурсы, то есть результат верификации полностью аналогичен результату метода АС. Таким образом, каждое требование обязательно получит вердикт и после верификации всех разбиений алгоритм ДАС завершается, предоставив результат, аналогичный

методу АС.

2. Шаг индукции. Пусть для R=k-1 (1<k<N) утверждение теоремы верно, докажем его для R=k.

Согласно алгоритму ДАС для каждого из верифицируемых разбиений возможно 3 варианта:

- Доказана корректность всех требований. С одной стороны, из этого следует корректность каждого из требований. С другой стороны, при удалении любого из данных требований также будет доказана их корректность.
- Найдено нарушение требований. Из этого следует нарушение данных требований в методе АС. Аналогично и при комбинации нарушенных автоматов с любыми другими могут быть найдены их нарушения. Если для всех остальных требований не была доказана корректность, то все они остаются во множестве проверяемых требований и будут использоваться при построении нового множества разбиений с R=k-1, для которого справедливо предположение индукции.
- Верификация требований исчерпала выделенные ресурсы. Если данное разбиение состояло только из одного требования, то данный результат аналогичен тому, что в методе АС данное требование исчерпывает ресурсы. В противном случае все требования остаются во множестве проверяемых и будут использоваться при построении нового множества разбиений с R=k-1, для которого справедливо предположение индукции.

Базовая стратегия разбиения для метода декомпозиции автоматной спецификации была предложена на основе понятия релевантности требования. Требование называется релевантным, если соответствующий ему автомат использовался при построении абстракции программы (только в этом случае требование потенциально может быть нарушено). Данная стратегия разбиения нацелена на поэтапное нахождение всех релевантных требований, далее на

совместную верификацию всех пока еще нерелевантных требований и последовательную верификацию (то есть не верификацию композиции требований) всех релевантных требований. Основная идея стратегии заключается в том, что нерелевантные требования целесообразно проверять совместно, так как они не ведут к экспоненциальному росту числа состояний, а релевантные к этому могут привести.

Метод декомпозиции автоматной спецификации можно рассматривать как расширение метода условной многоаспектной верификации, при этом метод декомпозиции автоматной спецификации позволяет использовать не только CEGAR, но и другие методы статической верификации. Кроме того, метод декомпозиции автоматной спецификации понижает сложность задач достижимости, что положительно сказывается на требованиях к ресурсам.

В **четвертой главе** «Реализация предложенных методов» описывается реализация предложенных методов на основе системы верификации LDV Tools и статического верификатора CPAchecker.

Расширения системы верификации были нацелены на то, чтобы появилась возможность подготавливать задачу достижимости на основе нескольких требований, формализованных как с помощью методов инструментирования, так и с помощью автоматных спецификаций, и решать ее с помощью любых из предложенных методов (обнаружение всех однотипных нарушений, условная многоаспектная верификация, автоматная спецификация и декомпозиция автоматной спецификации). Имеющиеся в системе LDV Tools модели требований были приведены в соответствие с ограничениями на инструментирование исходного кода программы, а кроме того, для каждого из требований была написана эквивалентная модель, формализованная с помощью метода автоматных спецификаций (описаны в Приложении А).

В инструменте CPAchecker был реализован метод условной многоаспектной

верификации как расширение подхода CEGAR и алгоритм декомпозиции автоматной спецификации на основе адаптивного статического анализа и наблюдательных автоматов. Для этого потребовалось модернизировать алгоритм CEGAR и разработать новый вид адаптивного анализа. Необходимая поддержка метода обнаружения всех однотипных нарушений в инструменте присутствовала (возможность продолжать верификацию после нахождения нарушения требования).

Для объединения преимуществ разработанных методов верификации композиции требований была предложена их последовательная комбинация, которая использует имеющуюся реализацию методов. Последовательная комбинация состоит в следующем: вначале проверяются все требования с помощью ускоренного за счет эвристик метода условной многоаспектной верификации с относительно небольшим ограничением на ресурсы. Если задача не может быть решена на первом шаге, то она решается с помощью более общего метода декомпозиции автоматной спецификации. Такая последовательная комбинация позволяет максимально использовать преимущества обоих методов в предложенной реализации. Хотя в наихудшем случае можно ожидать удвоение времени верификации, эксперименты на реальных программах показали существенное ускорение верификации (эксперименты описаны в пятой главе).

Все предложенные в данной работе методы были реализованы как расширения системы верификации LDV Tools и статического верификатора CPAchecker, и их реализации являются частью открытых (open source) проектов.

В **пятой главе** «Экспериментальная оценка предложенных методов» описываются результаты экспериментов со всеми предложенными методами. Для экспериментов использовались 30 требований системы верификации LDV Tools и все модули ядра операционной системы Linux версии 4.0-rc1. Для решения проблемы усложнения задач достижимости в методе условной многоаспектной

верификации также рассматривалось разбиение 30 требований на 2 группы требований.

Методы, нацеленные на обнаружение всех однотипных нарушений, позволили выявлять примерно в 1.5 раза больше нарушений требований (среди которых число реальных ошибок также увеличилось примерно в 1.5 раза), однако для этого потребовалось на 10% больше вычислительных ресурсов и несколько дней для ручного анализа результатов. Метод автоматных спецификаций по сравнению с инструментированием ускорил процесс верификации примерно на 10% и при этом смог успешно решить на 0.35% задач больше. Методы, нацеленные на верификацию композиции требований, продемонстрировали существенное ускорение всего процесса верификации при минимальных потерях результата (таблица 1). Зависимость ускорения методов верификации композиции требований относительно последовательной верификации от числа требований представлена на рисунке 1, зависимость процента потерь от числа требований – на рисунке 2 (для данного эксперимента 30 требований были разбиты по всем типам нарушений требований, в результате чего число требований было увеличено до 88).

На основе проведенных экспериментов можно сделать вывод о том, что предложенные методы статической верификации программного обеспечения являются достаточно перспективными и могут быть полезны на практике.

Метод условной многоаспектной верификации позволяет существенно повысить производительность верификации композиции требований с незначительным ухудшением результата. В рамках подхода CEGAR данный метод является наиболее производительным и масштабируемым. При проверке относительно большого числа требований разбиение проверяемой спецификации на группы требований позволяет улучшить характеристики метода для решаемых задач, что может использоваться в том случае, если заранее известны требования,

проверка которых часто ведет к чрезмерному росту числа состояний (такой сценарий использования возможен при проведении регрессионной верификации).

Метод обнаружения всех однотипных нарушений способен находить больше реальных ошибок в программах, чем последовательная верификация, однако для этого требуется больше ресурсов (как на саму верификацию, так и на ручной анализ результата). Метод условной многоаспектной верификации с обнаружением всех однотипных нарушений способен решать ту же задачу для композиции требований существенно быстрее метода обнаружения всех однотипных нарушений с незначительным ухудшением результата.

Метод автоматных спецификаций расширяет наблюдательные автоматы для формализации более сложных требований и в перспективе может стать полноценной заменой инструментированию. Данный метод может использоваться в последовательной верификации или в методе обнаружения всех однотипных нарушений.

Метод декомпозиции автоматной спецификации является расширением метода условной многоаспектной верификации за рамки подхода CEGAR на основе метода автоматных спецификаций. Данный метод является перспективным, поскольку может использоваться в любом подходе статической верификации и предоставляет удобный интерфейс для создания новых алгоритмов верификации композиции требований.

При использовании реализации предложенных методов в системе верификации LDV Tools рекомендуется использовать последовательную комбинацию методов, которая объединяет их преимущества с учетом решаемых задач в рамках подхода CEGAR и нацелена на минимизацию потерь результата при достаточно эффективном решении задач.

Метод	*Safe*	*Unsafe*	Потери / новые (%)	Реальные ошибки	Процессорное время / общее ускорение	
					CPAchecker	LDV Tools
Последовательная верификация (базовый метод)	118 703	667	–0 +0	121 –0 +0	3 889 000 1	6 742 000 1
Условная многоаспектная верификация	117 162	634	–1.36 +0.06	114 –9 +2	1 289 000 3.02	1 514 000 4.45
Условная многоаспектная верификация (с разбиением)	117 628	660	–0.93 +0.04	121 –2 +2	1 041 000 3.74	1 382 000 4.88
Декомпозиция автоматной спецификации	118 386	673	–0.45 +0.2	120 –2 +1	1 373 000 2.83	1 550 000 4.35
Последовательная комбинация	118 679	695	–0.26 +0.26	125 –0 +4	1 367 000 2.84	1 592 000 4.23

Таблица 1. Результаты сопоставления различных методов верификации композиции требований. Содержание колонок:

«*Safe*» – число задач, в которых доказана корректность программы относительно требования;

«*Unsafe*» – число задач, где найдено нарушение требования в программе;

«реальные ошибки» – число реальных ошибок среди результатов *Unsafe* (для них показаны изменения относительно базового метода последовательной верификации);

«потери» – процент потерь в сравнении с базовым методом;

«новые» – процент задач, которые не могли быть решены базовым методом, но были успешно решены с помощью предложенного метода;

«CPAchecker» – время для решения задач достижимости с помощью верификатора CPAchecker;

«LDV Tools» – общее время на подготовку и решение задач достижимости с помощью системы верификации LDV Tools, то есть для всего процесса верификации.

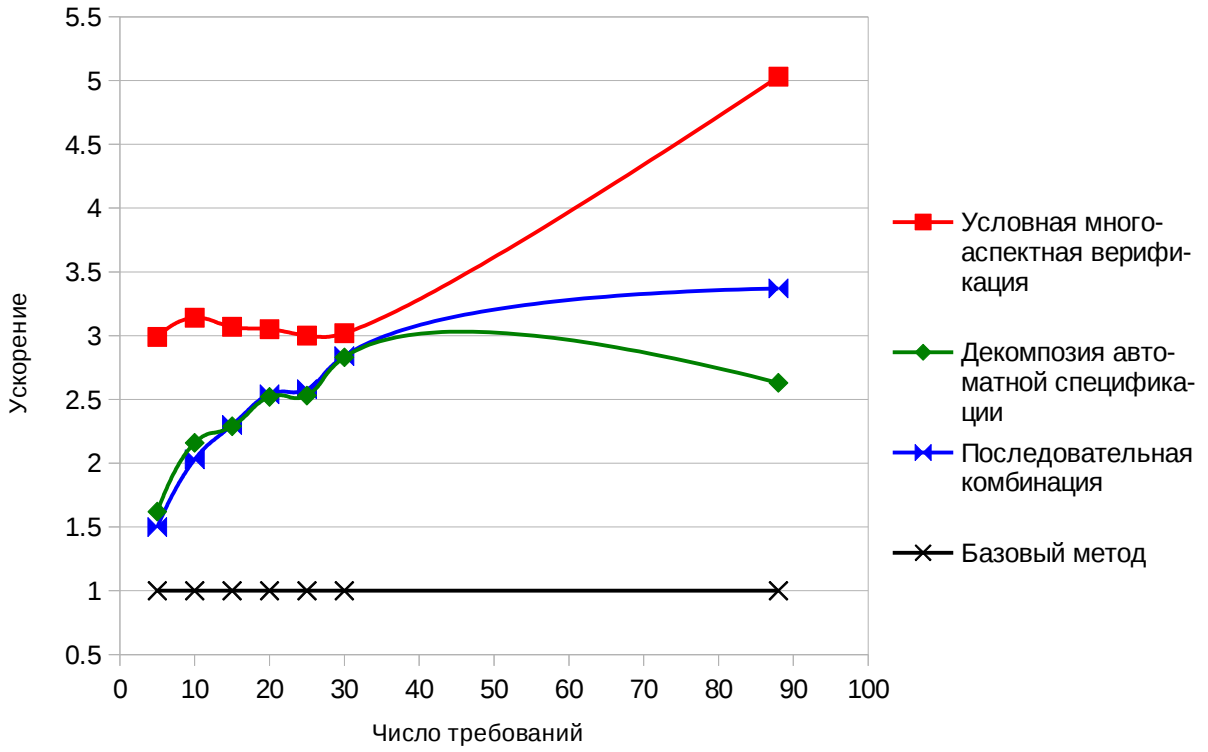

Рисунок 1. Зависимость ускорения методов верификации композиции требований относительно базового метода последовательной верификации от числа требований.

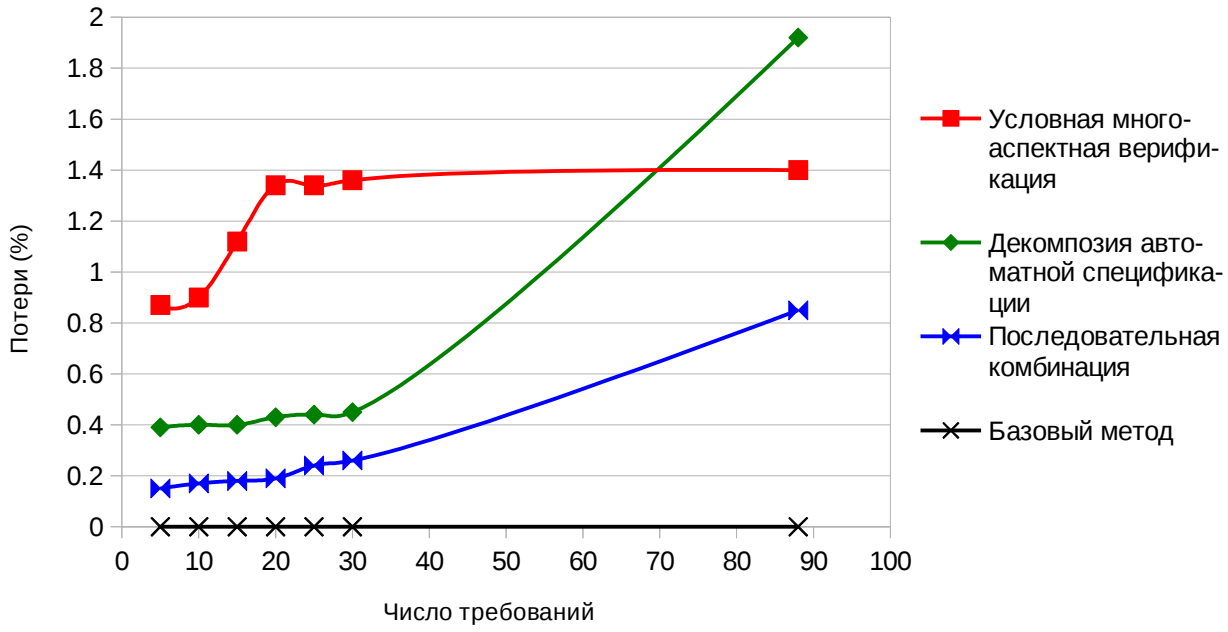

Рисунок 2. Зависимость процента потерь методов верификации композиции требований относительно базового метода последовательной верификации от числа требований.

Проведенные оценки характеристик предложенных методов показывают, что все методы приводят к сокращению времени верификации. Выбор того или иного метода или их комбинации зависит от конкретных условий процесса верификации. В отношении предложенных методов верификации композиции требований (условная многоаспектная верификация и декомпозиция автоматной спецификации) показано, что они повышают производительность верификации в 4-5 раз при незначительных потерях результата (примерно 1%) относительно базового, последовательного метода верификации. Метод обнаружения всех однотипных нарушений способен выявлять в 1.5 раза больше реальных ошибок при увеличении вычислительных ресурсов примерно на 10%, при этом метод требует проведения ручного анализа результатов.

В **заключении** перечисляются основные результаты диссертационной работы, выносимые на защиту, и возможные направления дальнейших исследований.

В **Приложении А** приведен перечень 30 требований системы LDV Tools, выполнимость которых проверялась в экспериментах. Для каждого требования представлено описание и его реализация с помощью метода автоматных спецификаций.

В **Приложении Б** приведены рекомендации по выбору параметров использования предложенных методов верификации.

В **Приложении В** приведены доказательства теорем и утверждений.

Работы автора по теме диссертации

1. Мордань В. О. Многоаспектная верификация модулей ядра операционной системы Linux // Материалы XXI Международной молодежной научной конференции студентов, аспирантов и молодых ученых «Ломоносов». – 2014. – С. 122-123.

2. Mordan V., Novikov E. Minimizing the number of static verifier traces to reduce time for finding bugs in Linux kernel modules // Proceedings of the 8th Spring/Summer Young Researchers' Colloquium on Software Engineering (SYRCoSE 2014). – 2014.
3. Mordan V., Mutilin V. Checking several requirements at once by CEGAR // Lecture Notes in Computer Science. – 2016. – Vol. 9609. – P. 218-232.
4. Мордань В. О., Мутилин В. С. Проверка нескольких требований за один запуск инструмента статической верификации с помощью CEGAR // Программирование. – 2016. – № 4. – С. 50-68.
5. Apel S., Beyer D., Mordan V., Mutilin V., Stahlbauer A. On-The-Fly Decomposition of Specifications in Software Model Checking // Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 349-361, 2016.

Свидетельства о государственной регистрации программы для ЭВМ

1. Мордань В.О. «Программный компонент для выявления нескольких ошибок в программном обеспечении». Свидетельство о государственной регистрации программы для ЭВМ № 2016616600 от 15.06.2016.
2. Мордань В.О. «Программный компонент для проверки нескольких правил корректности за один запуск инструмента статической верификации». Свидетельство о государственной регистрации программы для ЭВМ № 2016616661 от 16.06.2016.