На правах рукописи

Захаров Илья Сергеевич

Методы декомпозиции систем и моделирования окружения программных модулей для верификации Си-программ

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

АВТОРЕФЕРАТ
диссертации на соискание ученой степени
кандидата физико-математических наук

Москва – 2019

Работа выполнена в Федеральном государственном бюджетном учреждении науки «Институт системного программирования им. В.П. Иванникова Российской академии наук».

Научный руководитель: **Петренко Александр Константинович**
доктор физико-математических наук, профессор

Официальные оппоненты: **Тормасов Александр Геннадьевич**
доктор физико-математических наук, профессор, ректор автономной некоммерческая организации высшего образования «Университет Иннополис»

Буренков Владимир Сергеевич
кандидат технических наук, разработчик-исследователь акционерного общества «Лаборатория Касперского»

Ведущая организация: Федеральное государственное учреждение «Федеральный исследовательский центр Институт прикладной математики им. М.В. Келдыша Российской академии наук»

Защита состоится «23» мая 2019 года в 15:00 на заседании диссертационного совета Д 002.087.01 при Федеральном государственном бюджетном учреждении науки «Институт системного программирования им. В.П. Иванникова Российской академии наук», расположенном по адресу: 109004, Москва, ул. Александра Солженицына, 25.

С диссертацией можно ознакомиться в библиотеке и на сайте Федерального государственного бюджетного учреждения науки «Институт системного программирования им. В.П. Иванникова Российской академии наук».

Автореферат разослан «____» _____ 2019 г.

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

Общая характеристика работы

Актуальность темы исследования.

Вопросам надежности и информационной безопасности в современном мире уделяется все больше внимания. Одним из условий обеспечения надежности и защищенности операционных систем, систем управления базами данных, вебсерверов и других крупных программ на языке программирования Си, размер которых может достигать сотен тысяч и миллионов строк кода, является их верификация на разных этапах жизненного цикла. Особенно остро стоит задача верификации крупных программных систем ответственного назначения, используемых в органах государственной власти, транспортной, космической, авиационной и других отраслях.

Подходы, основанные на формальных методах верификации программ, до последнего времени из-за низкой масштабируемости применялись либо к моделям, разрабатываемым на языках формальной спецификации, либо к небольшим программам. На сегодняшний день уровень развития методов и инструментов уже позволяет выполнять проверку отдельных модулей программных систем на языке программирования Си. Работы по внедрению таких методов в промышленность особенно в отраслях, связанных с созданием программных систем ответственного назначения, постепенно выходят из фазы экспериментов и уже включаются в требования международных стандартов, например, ISO/IEC 15408 (Common Criteria), ISO/IEC 15408-1:2009, а также отраслевых стандартов в сфере транспорта ISO 26262-6:2018, EN50128, IEC 62279:2015 и авионики DO-178C. Для последнего из упомянутых стандартов было разработано отдельное приложение DO-333, посвященное применению формальных методов.

Одним из направлений верификации программ на основе формальных методов является проверка требований к модулям программной системы при помощи метода верификации моделей (англ. software model checking). Данный подход заключается в автоматическом построении модели программы на основе ее исходного кода, а затем ее формальной верификации на соответствие некоторому требованию. Главным достоинством такого подхода на практике является возможность обнаруживать ошибки, которые существенно снижают надежность и информационную безопасность всей системы в целом и которые

в рассматриваемых модулях трудно выявить при помощи других подходов к верификации программ. К таким ошибкам относятся, например, некорректная работа с памятью, переполнение целочисленного типа, гонки по данным в параллельных программах, некорректное использование определенного программного интерфейса.

Активное развитие методы верификации моделей программ получили благодаря работам Рупака Маджумдара и Дирка Бейера, авторам одного из первых инструментов верификации Си-программ BLAST; исследователей компании Майкрософт, разработавших и внедривших серию инструментов SLAM, SLAM2, Yogi и Q; исследовательской группе Оксфордского университета, развивающей проект CBMC под руководством Даниеля Кроенинга. Высокая трудоемкость работы инструментов верификации моделей программ все еще не позволяют применять данный подход к программным системам, размер которых приближается и даже превышает миллионы строк на языке программирования Си. Для автоматизированного применения инструментов были построены системы верификации SDV, LDV Tools и DC2, но реализованные в них методы применимы только к специализированным программам таким, как драйверы и программное обеспечение встраиваемых систем. Поэтому развитие данного метода верификации крупных программных комплексов является актуальным.

Для преодоления ограничений существующих в настоящее время систем верификации моделей Си-программ в случае, когда размеры проверяемой программной системы достигают нескольких сотен тысяч или миллионов строк кода, в данной работе предлагается следующее: перед верификацией провести декомпозицию системы на модули, учитывая ограничения как самих инструментов верификации; затем обеспечить условия применимости инструментов верификации, а именно: подготовить модели окружения модулей, поскольку инструмент верификации принимает на вход структурно полную программу на языке Си; далее, для того чтобы время верификации было приемлемым на практике, нужно использовать средства распараллеливания процесса верификации, которые доступны на современных многоядерных и распределенных вычислительных системах. В совокупности развитие методов и систем верификации в перечисленных направлениях позволит сократить трудоемкость и время верификации программ.

Цели и задачи работы.

Цель работы — развитие методов верификации крупных программных систем на языке программирования Си с использованием инструментов верификации моделей программ при помощи автоматизации декомпозиции системы на модули и синтеза моделей их окружения для сокращения трудоемкости и сроков верификации.

Для достижения поставленной цели были решены следующие задачи:

1. Исследовать существующие подходы к применению инструментов верификации моделей программ для проверки требований к Си-программам и выявить проблемы, препятствующие расширению применения этих подходов на практике.

2. Разработать архитектуру системы верификации Си-программ, выполняющей генерацию и решение набора верификационных задач при помощи автоматизированной декомпозиции системы на модули и синтеза моделей окружения модулей.

3. Разработать метод автоматизированной декомпозиции Си-программ на модули.

4. Разработать метод автоматизированного синтеза моделей окружения модулей, позволяющий адаптировать процесс синтеза для проверки разных видов требований и программ.

5. Реализовать систему верификации моделей Си-программ на основе разработанных архитектуры и методов, оценить реализацию системы верификации на практике.

Научная новизна.

Научной новизной обладают следующие результаты работы:

- Метод автоматизированной декомпозиции Си-программ на модули.

- Метод спецификации моделей окружения модулей Си-программ на основе композиции систем переходов и доказательство теоремы об изоморфизме множеств достижимых ошибочных состояний спецификации и результата ее трансляции на язык Си.

- Метод автоматизированного синтеза моделей окружения модулей, позволяющий адаптировать процесс синтеза для проверки разных видов требований и программ.

Теоретическая и практическая значимость.

Предложенные в диссертации методы реализованы в системе верификации Klever для проверки требований к программным системам на языке программирования Си с расширениями GNU при помощи инструментов верификации моделей программ, разрабатываемой в Институте системного программирования им. В.П. Иванникова РАН. Klever используется при проверке требований к модулям ядра операционной системы Linux, в которых при помощи данной системы верификации было найдено более ста ошибок. Система верификации используется для поиска дефектов и уязвимостей по безопасности, а также может применяться для проведения сертификационных исследований. Изложенные в данной работе результаты применяются в образовательном процессе на факультете ВМК МГУ и будут полезны для развития методов верификации моделей программных систем.

Методология и методы исследования.

Результаты диссертации были получены с использованием методов и моделей, применяемых при проведении верификации моделей программ. Математическую основу данной работы составляют теория графов, теория множеств, математическая логика и системы переходов.

Положения, выносимые на защиту.

- Метод автоматизированной декомпозиции Си-программ на модули.
- Метод спецификации моделей окружения модулей на основе композиции систем переходов.
- Метод автоматизированного синтеза моделей окружения модулей, позволяющий адаптировать процесс синтеза для проверки разных видов требований и программ.

На основе предлагаемых методов была реализована система верификации Klever, предназначенная для проверки требований к программным системам на

языке программирования Си с расширениями GNU методом верификации моделей программ. Продемонстрировано, что разработанная архитектура системы верификации позволяет проводить верификацию на различных многоядерных и распределенных вычислительных системах.

Степень достоверности и апробация результатов.

Основные результаты работы обсуждались на конференциях:

- 55-я научная конференция МФТИ (г. Москва, 2012 год).
- Международная научная конференция студентов, аспирантов и молодых учёных «Ломоносов-2013» (г. Москва, 2013 год).
- 7-й весенний/летний коллоквиум молодых исследователей в области программной инженерии (SYRCoSE, г. Казань, 2013 год).
- 10-я конференция разработчиков свободных программ (OSSDEVCONF, г. Калуга, 2013 год).
- Международная Ершовская конференция по информатике (PSI: Perspectives of System informatics, г. Санкт-Петербург, 2014 год).
- 5-й международный семинар Linux Driver Verification (г. Москва, 2015 год).
- 10-я летняя школа Microsoft Research (г. Кембридж, Великобритания, 2015 год).
- 1-й международный семинар, посвященный инструменту CPAchecker (г. Пассау, Германия, 2016 год).
- 2-й международный семинар, посвященный инструменту CPAchecker (г. Падерборн, Германия, 2017 год).
- 5-я Международная научно-практическая конференция «Инструменты и методы анализа программ» (г. Москва, 2017 год).
- Международная Ершовская конференция по информатике (PSI: Perspectives of System informatics, г. Москва, 2017 год).
- 5-я научно-практическая конференция OS DAY (г. Москва, 2018 год).

- Научно-практическая открытая конференция ИСП РАН им. В.П. Иванникова (г. Москва, 2018 год).

- Семинар ИСП РАН (г. Москва, 2018 год).

Публикации и зарегистрированные программы.

По теме исследования опубликовано 11 работ [1–11] из них 9 в изданиях перечня ВАК [2–10] и 6 входят в международную систему цитирования Scopus [4–6, 8–10]. В совместных работах [3, 4] личный вклад автора заключается в описании метода моделирования окружения, а в работах [2, 5, 6] в описании реализации метода моделирования окружения в системе верификации LDV Tools. В совместных работах [7, 9, 10] личный вклад автора состоит в описании методов декомпозиции программ на модули, а также спецификации и синтеза моделей окружения.

В ходе выполнения работы было получено 4 свидетельства о государственной регистрации программ для ЭВМ:

1. Свидетельство о государственной регистрации программы для ЭВМ № 2015662948: «Программный компонент решения задач верификации посредством использования инфраструктуры облачного сервиса».

2. Свидетельство о государственной регистрации программы для ЭВМ № 2017660773: «Klever Linux Kernel Verification Objects Generator».

3. Свидетельство о государственной регистрации программы для ЭВМ № 2017660774: «Klever Environment Model Generator for Linux Kernel Modules».

4. Свидетельство о государственной регистрации программы для ЭВМ № 2017660776: «Klever Verification Scheduler».

Личный вклад автора.

Все представленные в диссертации результаты получены лично автором.

Структура и объем диссертации.

Диссертация состоит из введения, обзора литературы, 5 глав, заключения и библиографии. Общий объем диссертации 157 страниц, включая 10 рисунков и 14 таблиц. Библиография включает 143 наименования на 18 страницах.

Содержание работы

Во введении обоснована актуальность диссертационной работы, сформулирована цель и аргументирована научная новизна исследований, показана практическая значимость полученных результатов.

В **первой главе** приведен обзор работ в области проверки требований к Си-программам при помощи метода верификации моделей.

Рассматривается подход проверки требований к модулям Си-программ. Данный подход реализован в таких инструментах верификации моделей программ, как CPAchecker, CBMC, SLAM2, Yogi. В 2012 году было организовано сообщество разработчиков инструментов верификации моделей программ SV-COMP, которое на сегодня насчитывает порядка тридцати групп исследователей и инженеров. Данным сообществом были предложены единые форматы входных и выходных данных и вспомогательные средства для использования различных инструментов верификации моделей на практике.

Инструменты верификации моделей Си-программ принимают на вход *верификационные задачи*. Верификационная задача содержит препроцессированный исходный код некоторого модуля программы, размер которого не превышает нескольких десятков тысяч строк кода на языке программирования Си, спецификацию свойства корректности, конфигурационные параметры и ограничения на вычислительные ресурсы. Инструменты верификации поддерживают небольшой набор свойств корректности таких, как, например, корректность работы с памятью, недостижимость ошибочного оператора, завершимость программы. На практике задачу проверки определенного требования к программе сводят к проверке одного или нескольких поддерживаемых свойств корректности за счет разработки модели требования на языке программирования Си, которая добавляется к исходному коду верификационной задачи. Результат решения верификационной задачи состоит из вердикта и свидетельства коррект-

Рис. 1: Схема решения верификационных задач.

ности или нарушения. Вердикт может принимать значения **True**, если программа корректна, **False** при обнаружении нарушения свойства корректности или **Unknown**, например, если для верификации не хватило вычислительных ресурсов. Свидетельства корректности или нарушения предназначены для экспертизы пользователем или автоматической валидации. Схема работы инструментов верификации моделей программ изображена на рисунке 1.

Для выполнения верификации крупной программной системы требуется декомпозировать исходный код на модули размером не более нескольких десятков тысяч строк на языке программирования Си и формализовывать требования. Чтобы достичь результатов с низким числом пропущенных ошибок и ложных предупреждений об ошибках, для каждого модуля программы следует подготовить модель его окружения. Модель окружения дополняет модуль до структурно-полной программы с единственной точкой входа (main), а также содержит модели релевантных проверяемому требованию неопределенных функций, вызываемых в модуле. Анализ данных работ показал, что главным ограничением при применении инструментов верификации моделей программ является высокая трудоемкость подготовки верификационных задач.

Далее рассматриваются подходы автоматизации применения инструментов верификации моделей программ к драйверам операционных систем (ОС) и встраиваемому программному обеспечению. Данные подходы реализованы в системах верификации Microsoft SDV, LDV Tools и DC2, которые показали возможность как получения результатов с высокой точностью, так и существенного снижения необходимых трудозатрат при верификации. Системы автоматизируют подготовку верификационных задач, включая самые трудоемкие этапы: декомпозицию программ на модули, синтез моделей окружения и требований.

В таблице 1 представлен результат сравнительного анализа существующих систем верификации. В системе верификации DC2 декомпозиция не выполняется, а SDV и LDV позволяют выделять в качестве модулей только драйверы

Критерий сравнения	SDV	LDV	DC2
Поддержка адаптивной декомпозиции программ	✗	✗	✗
Наличие средств для разработки или исправления моделей окружения	✓	✗	✗
Синтез моделей окружения	✗	✓	✓
Параллельные генерация и решение верификационных задач	✗	✗	✗

Таблица 1: Сравнение систем верификации моделей программ.

без возможности корректировки состава модулей. Разработчики системы верификации SDV подготовили модели окружения драйверов определенных видов на языке Си вручную без использования методов автоматизации. Подход к автоматической генерации моделей окружения, реализованный в системе верификации LDV Tools, опирается на эвристические предположения и не позволяет повысить точность моделей окружения, что приводит к увеличению числа ложных предупреждений об ошибках. Метод генерации моделей окружения, реализованный в системе верификации DC2, предназначен только для замены определений функций программы на упрощенные аннотации, что не позволяет описать вызов точек входа в таких моделях окружения. Важным ограничением систем верификации является отсутствие средств для распараллеливания генерации и решения верификационных задач, из-за чего время верификации может превышать часы и даже дни.

Затем описываются работы в области распараллеливания алгоритмов верификации моделей. Для верификации моделей Си-программ в данной работе отдается предпочтение методам параллельного решения верификационных задач с использованием вычислительных кластеров, построенных на основе IaaS (англ. Infrastructure as a Service) платформ и специализированных облачных сервисов.

В конце главы формулируется ряд основных требований к системам верификации моделей Си-программ, которые должны автоматизировать процесс генерации и решения верификационных задач. Декомпозицию и синтез моделей окружения следует автоматизировать таким образом, чтобы пользователь имел возможность выполнять адаптацию данных процессов для проверки разных требований и программ. Адаптацию предлагается выполнять при помощи

конфигурирования, а синтез моделей окружения производить на основе спецификаций предположений об окружении. Важными требованиями к системе верификации являются возможность использования распределенных вычислительных систем и обеспечение воспроизводимости результатов.

Во **второй главе** представлены три основных метода, предложенных в данной работе: метод автоматизированной декомпозиции Си-программ на модули, метод спецификации моделей окружения модулей на основе композиции систем переходов и метод автоматизированного синтеза моделей окружения модулей.

В первом разделе главы вводится ряд ключевых определений. Под модулем программы будем понимать набор из нескольких файлов с исходным кодом на языке программирования Си. Назовем *событиями взаимодействия* модуля и окружения вызовы функций программного интерфейса модуля, включая его точки входа, а также обращения на чтение или запись к глобальным переменным и областям памяти, доступным окружению. *Сценарием взаимодействия* модуля и окружения называется совокупность последовательностей событий взаимодействия, возможных при реальном выполнении программы. Для последовательностей событий сценария взаимодействия можно сформулировать требования к порядку событий и зависимостям по данным между ними.

Моделью окружения называется вспомогательный исходный код на языке программирования Си, который реализует модели сценариев взаимодействия модуля и окружения, а также содержит модели необходимых неопределенных функций. Спецификацией предположений об окружении будем называть описание фрагментов модели на некотором языке предметной области.

Во втором разделе представлен *генератор верификационных задач*, структура которого изображена на рисунке 2. В качестве входных данных генератор получает конфигурационные параметры, спецификацию декомпозиции, разработанную пользователем для уточнения состава модулей программы, спецификации требований и предположений об окружении, а также базу сборки, в которой содержатся структурированные данные о процессе сборки программы и описание ее программного интерфейса.

Третий раздел рассматривает декомпозицию Си-программ. При декомпозиции определяется та часть исходного кода программы, которую необходимо

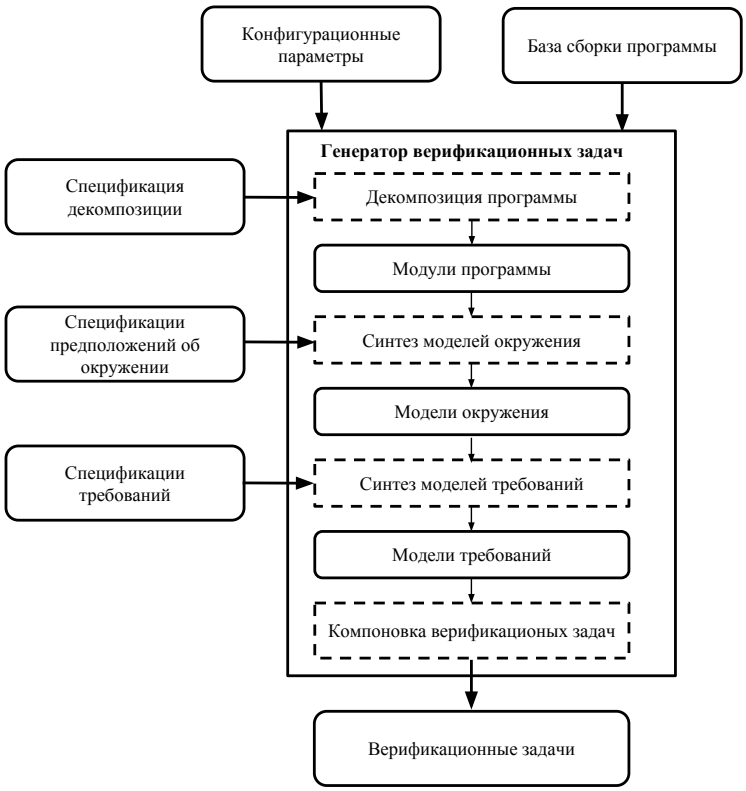

Рис. 2: Структура генератора верификационных задач.

верифицировать, и в ней выделяются модули для генерации верификационных задач.

Перед выполнением верификации оценить результат декомпозиции затруднительно, так как на сегодняшний день нет надежных методов прогнозирования возможности решения верификационной задачи при заданных ограничениях на вычислительные ресурсы. Поэтому декомпозицию предлагается выполнять автоматизированно с возможностью настройки процесса декомпозиции и коррекции состава модулей.

Метод автоматизированной декомпозиции программ на модули состоит в выполнении следующих шагов: определения файлов и функций в составе программы, выделения логических компонентов, выбора целевых модулей, коррекции состава модулей согласно спецификации декомпозиции и агрегации. Входными данными являются конфигурационные параметры, где пользователь указывает часть программы, которую требуется верифицировать, база сборки и *спецификация декомпозиции*, разработанная вручную пользователем для коррекции состава модулей, получаемых автоматически.

На шаге определения состава программы выполняется построение двух

ориентированных графов на основе базы сборки: графа файлов и графа функций. Вершины графов соответствуют файлам и функциям соответственно, а ребра отражают зависимости по вызову функций.

На следующем шаге выполняется деление множества вершин графа файлов на непересекающиеся подмножества, называемые модулями, согласно некоторой *стратегии выделения модулей*. Стратегия определяет специализированный алгоритм, предназначенный для выделения модулей в определенной программной системе. Результатом деления является граф модулей, в котором вершины соответствуют модулям, а ребра отражают зависимости между ними по вызову функций.

Следующим шагом является выбор модулей, которые требуется верифицировать. Такие модули называются целевыми и определяются при помощи конфигурационных параметров, в которых указаны пользователем целевые директории и файлы с исходным кодом, функции и модули. Соответствующие им вершины графов модулей и файлов на данном шаге помечаются как целевые.

Если пользователь задал спецификацию декомпозиции, то выполняется коррекция состава модулей и соответствующего графа.

На последнем шаге выполняется выбор наборов модулей в графе для каждого целевого модуля, минимизируя число точек входа совокупности модулей для сокращения трудоемкости моделирования для них окружения, согласно некоторой *стратегии агрегации*. Стратегия определяет специализированный алгоритм поиска наборов модулей в графе определенного вида. Каждый набор модулей становится новым модулем, для которого будут сгенерированы верификационные задачи.

Предложенный метод позволяет адаптировать процесс декомпозиции к разным видам Си-программ при помощи конфигурирования и разработки новых стратегий и спецификаций декомпозиции.

В четвертом разделе изложен метод спецификации моделей окружения модулей Си-программ.

Для модулей с единственной точкой входа предлагается разрабатывать модели неопределенных функций на языке программирования Си. Процесс моделирования окружения для библиотек и событийно-ориентированных программ требует построения композиции из моделей сценариев взаимодействия модуля

и окружения. Агрегация модулей программы позволяет снизить трудозатраты на моделирование, но при этом могут возникнуть новые требования к последовательностям событий разных сценариев взаимодействия, которые требуется учитывать при спецификации моделей окружения.

Метод спецификации модели окружения для некоторого модуля заключается в разработке *промежуточной модели окружения*, которая определяется парой $M = < P, E >$ и описывает модель окружения некоторой параллельной программы, использующей программный интерфейс управления потоками согласно стандарту POSIX. P состоит из множеств вспомогательных переменных, функций, макросов и типов, определенных на языке программирования Си. E содержит конечное множество моделей сценариев взаимодействия, которые делятся на два вида: *модели функций окружения* и *модели потоков окружения*. Каждая модель функции окружения реализует модель сценария взаимодействия, события которого происходят во время выполнения некоторой функции, определенной в окружении. Модель потока окружения реализует модель сценария взаимодействия, события которого выполняются в отдельном потоке.

Каждая модель сценария определяется системой переходов $\varepsilon = < \mathcal{V}, \mathcal{A}, \alpha_0, \mathcal{R} >$. Где $\mathcal{V}$ — это множество переменных состояния, заданных на языке Си, $\mathcal{A}$ — это множество *действий*, $\alpha_0 \in \mathcal{A}$ называется начальным действием, а $\mathcal{R} : \mathcal{A} \times \mathcal{A}$ — отношение переходов для задания ограничений на порядок следования действий. Будем различать действия трех видов: *прием сигнала*, *отправка сигнала* и *модель события*.

Каждая модель события описывается при помощи предусловия, постусловия и базового блока кода на языке программирования Си, состоящего из операций над переменными состояния и глобальными переменными модуля и модели окружения. Базовые блоки содержат вызовы точек входа модулей и инициализацию параметров для них. Действия передачи сигналов выполняются согласно модели синхронизации рандеву между парами моделей сценариев и служат для задания ограничений на порядок событий разных сценариев взаимодействия и на зависимости по данным между ними.

Промежуточная модель окружения компонуется вместе с исходным кодом модуля при помощи инструментации, поэтому семантика модели без модуля не

определена. Предложенные модели сценариев можно описывать на языке программирования Си. Но для описания действий посылки и получения сигналов предлагается расширить язык Си вспомогательными операциями для удобства спецификации. Поэтому промежуточные модели окружения перед верификацией модуля должны быть предварительно транслированы на язык программирования Си. Необходимо показать, что трансляция промежуточной модели окружения не влияет на возможность обнаружения ошибок в модуле методом верификации моделей. Для этого рассмотрен некоторый Си-подобный язык программирования LZ и его расширение ELZ, полученное добавлением операций пересылки сигналов. Языки сохраняют именно те особенности семантики программ на языке Си, на которые влияет добавление операций посылки и получения сигналов: порядок выполнения операций, синхронизацию потоков и передачу данных между ними, проверку логических выражений и целочисленную арифметику. Семантика выполнения программ на данных языках задана при помощи символических систем переходов.

Затем были проанализированы свойства предложенной семантики выполнения программ на языках LZ и ELZ, сформулированы и доказаны вспомогательные леммы и основная теорема:

Лемма 1. *Истинность предиката пути, на котором могут выполняться подряд две независимые операции не зависит от порядка выполнения данных операций между собой.*

Лемма 2. *На эквивалентность блоков не влияют независимые операции выполняемые параллельно в других процедурах.*

Лемма 3. *Предикат пути через блоки кода с захваченной блокировкой принимает истинное значение только в том случае, если на этом пути операции между метками входа и выхода каждого блока под блокировкой может одновременно выполнять только один поток.*

Лемма 4. *На пути выполнения блока из операций $flag$ и $recv$ приема сигнала d всегда происходит посылка данного сигнала из другого потока.*

Теорема 1. *Множества достижимых ошибочных состояний исходной программы на языке ELZ и результата ее трансляции на язык LZ изоморфны.*

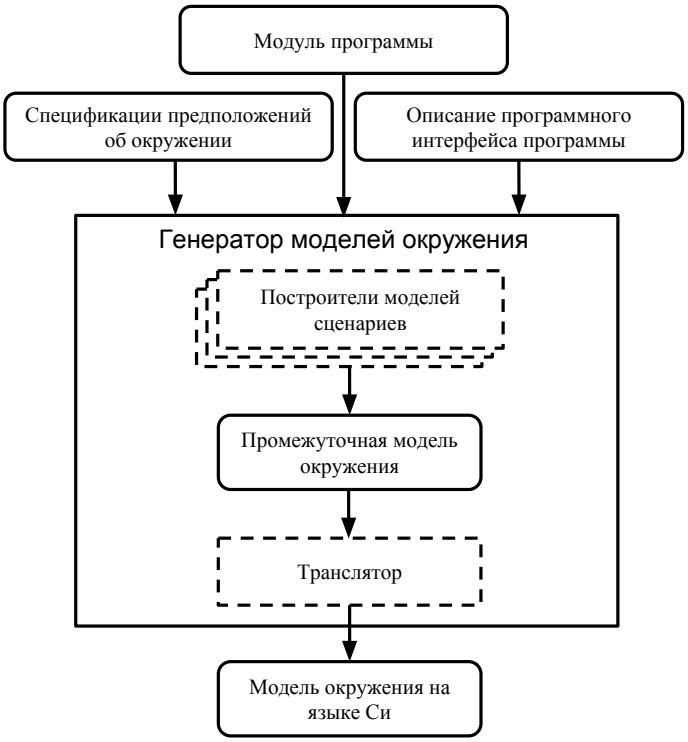

Рис. 3: Структура генератора моделей окружения.

Результат теоремы справедлив при проверке требований методом верификации моделей к программе на языке Си, сведенных к доказательству недостижимости ошибочного оператора, в предположениях, что программа использует только интерфейс управления потоками POSIX и инструмент при верификации строит модель с теми же свойствами, которыми обладают рассмотренные при доказательстве теоремы модели программ на языках LZ и ELZ.

В пятом разделе описан **метод автоматизированного синтеза моделей окружения на языке программирования Си**, который заключается в построении моделей сценариев взаимодействия на основе спецификаций предположений об окружении, подготовке промежуточной модели окружения как композиции моделей сценариев взаимодействия и ее трансляции на язык Си. Метод выполняется *генератором моделей окружения*, структура которого изображена на рисунке 3.

В зависимости от устройства верифицируемой программы используются несколько построителей моделей сценариев взаимодействия для промежуточной модели окружения. Каждый построитель выполняет подготовку моделей сценариев на основе соответствующих спецификаций предположений об окружении и описания программного интерфейса из базы сборки программы. Специ-

фикации предположений об окружении содержат модели отдельных сценариев взаимодействия на разных языках предметной области, расширяющих формат задания промежуточной модели окружения для упрощения описания моделей сценариев определенного вида.

Затем выполняется трансляция промежуточной модели окружения на язык программирования Си. Транслятор позволяет выполнять настройку процесса генерации модели окружения в зависимости от проверяемого требования и используемого инструмента верификации.

Шестой раздел описывает синтез моделей требований на основе спецификаций требований, разрабатываемых на аспектно-ориентированном расширении языка программирования Си, согласно методу, реализованному в системе верификации LDV Tools.

В седьмом разделе рассматривается процесс компоновки верификационных задач в формате, предложенном сообществом SV-COMP, из модулей программы и соответствующих синтезированных моделей окружения и требований. На практике на данном шаге выполняется препроцессирование, инструментация и построение среза исходного кода верификационной задачи (англ. slicing).

В **третьей главе** описана архитектура системы верификации моделей программ для генерации и решения верификационных задач с использованием распределенных вычислительных систем.

Структура системы верификации изображена на рисунке 4. Сервер предназначен для организации взаимодействия пользователя и компонентов системы верификации, а также для хранения данных. При помощи пользовательского интерфейса сервера пользователи формируют *верификационные задания*, состоящие из базы сборки программы, спецификаций и конфигурационных параметров. Генератор выполняет автоматическую подготовку верификационных задач для определенного верификационного задания согласно схеме, изложенной во второй главе. Инструменты верификации моделей программ решают верификационные задачи, подготовленные в рамках решения верификационных заданий. Решатель верификационных задач и заданий предназначен для управления вычислительными ресурсами и запуска экземпляров остальных компонентов.

Каждое верификационное задание и задача имеют максимальное ограничение на вычислительные ресурсы: объем доступной оперативной памяти и памя-

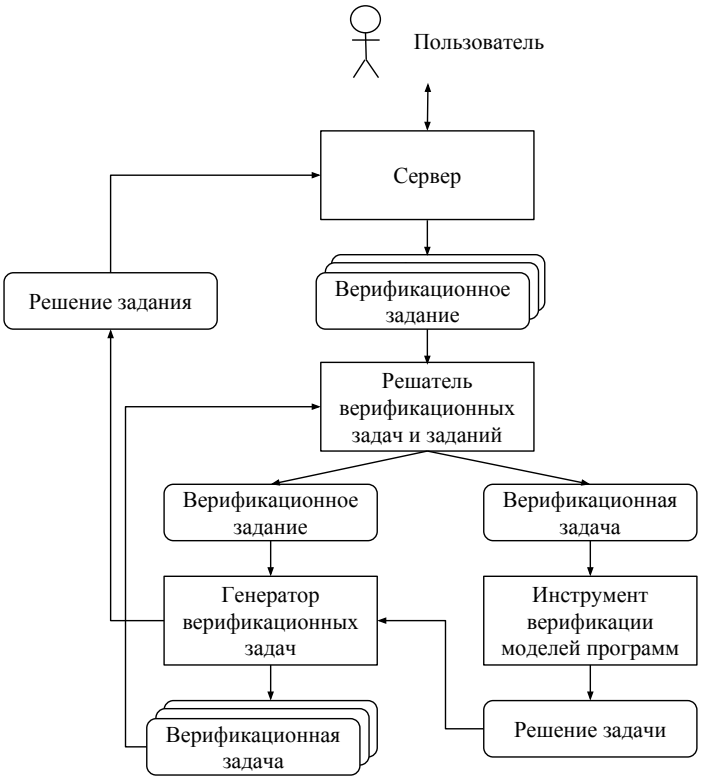

Рис. 4: Структура системы верификации моделей программ.

ти на диске, время выполнения и процессорное время. Ограничения необходимо соблюдать для получения воспроизводимых результатов верификации.

При получении на вход верификационного задания или задачи решатель выполняет процедуру планирования и запуска экземпляров необходимых компонентов, выполняемую специальными модулями. Модули предназначены обеспечить взаимодействие решателя с разными системами управления вычислительными кластерами и облачными сервисами.

В **четвертой главе** представлена реализация предложенных методов в системе верификации Klever.

Система верификации Klever является проектом с открытым исходным кодом и предназначена для проверки требований к программным системам на языке программирования Си с расширениями GNU при помощи инструментов верификации моделей программ.

Для программных систем BusyBox и ядра ОС Linux были разработаны специализированные стратегии выделения модулей. Для агрегации модулей были реализованы алгоритмы, опирающиеся на граф вызова функций и отчеты о покрытии модулей при верификации. В генераторе моделей окружения ре-

ализованы три построителя моделей сценариев, из которых два предназначены для верификации драйверов и подсистем ядра ОС Linux, а третий может быть использован для различных программ. В данном компоненте реализован конфигурируемый транслятор, поддерживающий построение последовательной или параллельной модели окружения. Синтез моделей требований выполнен с возможностью использования спецификаций требований, разработанных для системы верификации LDV Tools. Инструментация исходного кода в компоновщике реализована при помощи инструмента CIF (C Instrumentation Framework), основанного на компиляторе GCC версии 7.1. Построение среза исходного кода верификационной задачи выполняется при помощи CIL из набора инструментов по дедуктивной верификации Си-программ AstraVer Toolset.

Решатель реализован с поддержкой решения верификационных задач на одном вычислительном узле или в вычислительном кластере, управляемом при помощи VerifierCloud.

Система позволяет применять инструменты верификации моделей Си-программ, поддерживающие формат верификационных задач сообщества SV-COMP. На практике используются различные версии инструментов верификации CPAchecker и Ultimate Automizer.

В **пятой главе** представлены результаты практического применения предложенных методов, реализованных в системе верификации Klever, а также сделаны выводы о границах их практической применимости.

Предложенные в данной работе методы нацелены преимущественно на сокращение трудоемкости и скорости верификации, а уровень точности верификации не должен понизиться. Во втором разделе анализируются результаты экспериментов по верификации драйверов и подсистем ОС Linux и приводятся оценки улучшения характеристик, достигнутых при помощи разработанных методов.

На первом этапе выполнения экспериментов, чтобы оценить трудоемкость верификации для получения результатов с высоким уровнем точности, был выбран набор из 49 Serial драйверов ОС Linux версии 3.14.79, размер исходного кода которых составляет 30 тыс. строк кода. Затем рассматривается переход к верификации остальных драйверов ОС Linux, размер которых составляет более 3 млн. строк кода, и оцениваются дополнительные трудозатраты. На за-

ключительном этапе выполняется верификация подсистем без разработки новых спецификаций, чтобы подтвердить возможность повторного использования имеющихся артефактов. Оценка трудоемкости всех трех этапов представлена в таблице 2[1].

Для измерения времени верификации проверялись наборы из 49 Serial и 3864 других драйверов на соответствие 30 требованиям. Результаты, представленные в таблице 3, были получены с использованием 2-х физ. ядер, 4-х физ. ядер и вычислительного кластера из 30 узлов с 4-я физ. ядрами в каждом.

Уровень покрытия по функциям при верификации драйверов составил 100% для набора Serial и 45% для остальных драйверов. Чтобы достичь более высокого уровня необходимо затратить существенно больше усилий на разработку спецификаций предположений об окружении. В то же время при помощи системы верификации Klever уже было выявлено более ста ошибок в различных драйверах.

В таблице 4 представлены результаты сравнительного анализа вердиктов, полученных при верификации Serial драйверов при помощи систем верификации Klever и LDV Tools. Проверка требований корректности работы с памятью и отсутствия гонок по данным возможна только в системе верификации Klever благодаря конфигурируемости генератора моделей окружения, а остальные 30 проверяемых требований совпадают. Результаты демонстрируют, что система верификации Klever позволяет получить меньше ложных предупреждений об ошибках в связи с повышением уровня точности моделей окружения, а также позволяет выявить новые ошибки.

Чтобы выяснить, как на результаты верификации влияет проверка разных версий ядра ОС Linux с одним и тем же набором спецификаций, были рассмотрены подсистемы GPIO, TTY и CHAR версий ядра ОС Linux, начиная с 3.9 и заканчивая 3.19. На рисунке 5 представлено среднее число полученных вердиктов в зависимости от версии ядра для каждой подсистемы. Были проанализированы вручную 488 изменений в упомянутых подсистемах, чтобы выявить исправления ошибок, которые являются нарушениями формализованных в рамках системы верификации Klever требований. Среди 8 таких исправлений

[1] Разработка спецификаций требований выполнялась за рамками данной работы, а оценка трудоемкости приведена для полноты описания процесса подготовки к верификации.

Этап разработки	Serial Драйверы	Все драйверы	Подсистемы
Стратегий декомпозиции	0,25 чел. мес. 100LOC Python	0 чел. мес. 100LOC Python	0,25 чел. мес. 120LOC Python
Построителей моделей сценариев	3 чел. мес. 3KLOC Python	0 чел. мес. 3KLOC Python	0,5 чел. мес. 3,5KLOC Python
Спецификаций предположений об окружении	4,5 чел. мес. 7KLOC DSL	5,5 чел. мес. 17KLOC DSL	0 чел. мес. 17KLOC DSL
Спецификаций требований	6 чел. мес. 550LOC DSL	9 чел. мес. 1500LOC DSL	0,25 чел. мес. 1500LOC DSL
Итого	13,75 чел. мес.	14,5 чел. мес.	1 чел. мес.

Таблица 2: Трудоемкость верификации драйверов и подсистем ОС Linux.

Верификационное задание	2 физ. ядра	4 физ. ядра	30 * 4 физ. ядра
Serial драйверы (30KLOC)	5ч	2,7ч	0,5ч
Все драйверы (3MLOC)	600ч	195ч	11ч

Таблица 3: Время верификации драйверов ОС Linux версии 3.14.79.

ошибок, 4 могут быть успешно выявлены системой верификации, две ошибки не относятся к рассматриваемым конфигурации и архитектуре, а еще две были пропущены из-за ограничений инструмента верификации.

В третьем разделе исследуется возможность применения системы верификации Klever к пользовательским программам. Для этого выполнены эксперименты по верификации апплетов проекта BusyBox, которые являются пользовательскими преобразующими программами с единственной точкой входа. Трудоемкость подготовки к верификации представлена в таблице 5, а статистика по полученным вердиктами в таблице 6. Уровень покрытия по функциям составил 94%.

В последнем разделе делаются выводы о пределах применимости разработанных методов и системы верификации, а также делается заключение о целесо-

Верификационное задание	Ложное предупреждение	Ошибка	Нет вердикта
Klever, 32 требования	16	6	46
Klever, 30 требований	5	1	16
LDV Tools, 30 набор требований	31	0	9

Таблица 4: Вердикты при верификации Serial драйверов ОС Linux версии 3.14.79.

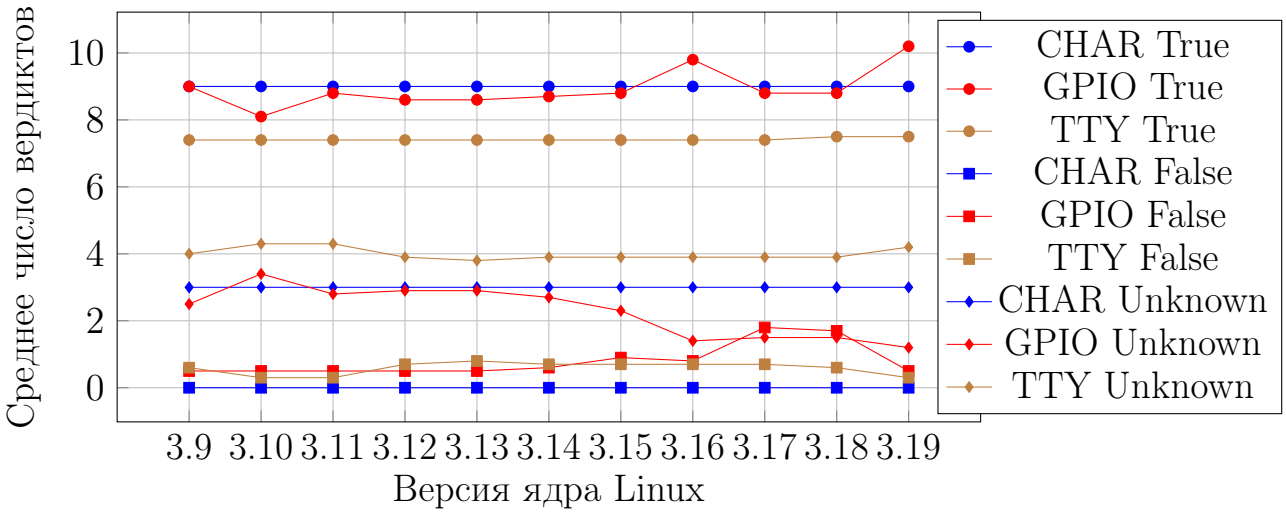

Рис. 5: Среднее число вердиктов при верификации целевых подсистем в зависимости от версии ОС Linux.

Разработка стратегий декомпозиции	Разработка построителей моделей сценариев	Спецификация предположений об окружении	Спецификация требований
0,25 чел. мес. 100LOC Python	0 чел. мес.	0,25 чел. мес. 200LOC DSL	0,5 чел. мес. 300LOC DSL

Таблица 5: Трудоемкость верификации апплетов проекта BusyBox.

Верификационное задание	Ложное предупреждение	Ошибка	Нет ошибок	Нет вердикта
BusyBox (200KLOC)	6	1	159	133

Таблица 6: Вердикты, полученные при верификации апплетов проекта BusyBox.

образности их использования при верификации программных систем, имеющих событийно-ориентированное архитектуру, например, операционных систем, веб-серверов и встраиваемого программного обеспечения.

В **Заключении** представлены основные результаты работы.

Основные результаты работы

Основные научные результаты, полученные в диссертационной работе и выносимые на защиту, состоят в следующем:

- Разработан метод автоматизированной декомпозиции Си-программ на модули.
- Разработан метод спецификации моделей окружения модулей на основе композиции систем переходов.
- Разработан метод автоматизированного синтеза моделей окружения модулей, позволяющий адаптировать процесс синтеза для проверки разных видов требований и программ.

На основе предлагаемых методов была реализована система верификации Klever, предназначенная для проверки требований к программным системам на языке программирования Си с расширениями GNU методом верификации моделей программ. Продемонстрировано, что разработанная архитектура системы верификации позволяет проводить верификацию на различных многоядерных и распределенных вычислительных системах.

Публикации автора по теме диссертации

1. Zakharov I., Khoroshilov A., Mutilin V., Novikov E. Generating environment model for Linux device drivers // Spring/Summer Young Researchers' Colloquium on Software Engineering. 2013. No. 7.
2. Захаров И., Мутилин В., Новиков Е., Хорошилов А. Моделирование окружения драйверов устройств операционной системы Linux // Труды Института системного программирования РАН. Т. 25. 2013. С. 85—112.

3. Захаров И., Мандрыкин М., Мутилин В., Новиков Е., Петренко А., Хорошилов А. Конфигурируемая система статической верификации модулей ядра операционных систем // Труды Института системного программирования РАН. Т. 26. 2014. С. 5—42.

4. Zakharov I., Mandrykin M., Mutilin V., Novikov E., Petrenko A., Khoroshilov A. Configurable toolset for static verification of operating systems kernel modules // Programming and Computer Software. 2015. Vol. 41, no. 1. P. 49–64.

5. Zakharov I., Mutilin V., Khoroshilov A. Pattern-based environment modeling for static verification of Linux kernel modules // Programming and Computer Software. 2015. Vol. 41, no. 3. P. 183–195.

6. Khoroshilov A., Mutilin V., Novikov E., Zakharov I. Modeling Environment for Static Verification of Linux Kernel Modules // Perspectives of System Informatics. Berlin, Heidelberg, 2015. P. 400–414.

7. Захаров И., Новиков Е. Инкрементальное построение спецификаций моделей окружения и требований для подсистем монолитного ядра операционных систем // Труды Института системного программирования РАН. Т. 29. 2017. С. 25—48.

8. Zakharov I. A Survey of High-Performance Computing for Software Verification // Tools and Methods of Program Analysis. Cham, 2018. P. 196–208.

9. Novikov E., Zakharov I. Towards Automated Static Verification of GNU C Programs // Perspectives of System Informatics. Cham, 2018. P. 402–416.

10. Novikov E., Zakharov I. Verification of Operating System Monolithic Kernels without Extensions // Proceedings of the International Symposium On Leveraging Applications of Formal Methods, Verification and Validation. 2018.

11. Novikov E., Zakharov I. Compositional Environment Modelling for Verification of GNU C Programs // Proceedings of the 2018 Ivannikov ISP RAS Open Conference. 2018. P. 39–46.