На правах рукописи

Мандрыкин Михаил Усамович

Моделирование памяти Си-программ для инструментов статической верификации на основе SMT-решателей

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

Автореферат
диссертации на соискание ученой степени кандидата физико-математических наук

Москва — 2016

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 Федерального государственного бюджетного учреждения науки Институт системного программирования Российской академии наук.

Официальные оппоненты: **Галатенко Владимир Антонович**, доктор физико-математических наук, заведующий сектором Федерального государственного учреждения "Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук".

Климов Юрий Андреевич, кандидат физико-математических наук, старший научный сотрудник Федерального государственного учреждения "Федеральный исследовательский центр Институт прикладной математики им. М.В. Келдыша Российской академии наук"

Ведущая организация: Федеральное государственное бюджетное образовательное учреждение высшего образования «Санкт-Петербургский государственный университет»

Защита состоится **“15” декабря 2016 г.** в **16 часов** 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 “_____”_____2016 г.

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

Актуальность

Язык программирования Си продолжает оставаться одним из наиболее широко используемых языков программирования в области системного программирования, в частности при разработке операционных систем, драйверов, сред окружения и поддержки времени выполнения для различных языков программирования, а так же при создании других систем, предъявляющих высокие требования к производительности или объему исполняемых программ. При этом одним из основных преимуществ использования языка Си для реализации высокопроизводительных систем является слабая статическая типизация и присутствие широкого набора доступных операций с указателями, позволяющие эффективно управлять использованием памяти, в том числе явно манипулируя адресами размещаемых в ней данных. Ко многим высокопроизводительным системам предъявляются среди прочих требования по высокой надежности, а в некоторых случаях — и по безопасности. Поэтому продолжает оставаться актуальной задача верификации Си-программ.

Эта задача может решаться с применением методов динамической, статико-динамической и статической верификации. В силу того, что каждый из подходов обладает своими преимуществами и ограничениями необходимо вести развитие всех этих направлений. Вместе с тем, только статическая верификация может дать доказательство отсутствия ошибок, по крайней мере некоторых классов ошибок, или дать доказательство полного соответствия программ заданным формальным спецификациям.

Методы статической верификации ранее показали свою применимость, в частности, для верификации модулей и отдельных подсистем в гипервизорах и ядрах операционных систем, что в свою очередь является важным аргументом, подтверждающим актуальность развития этих методов.

Область статической верификации программ в настоящее время активно развивается по всем основным направлениям, а именно развиваются автоматическая статическая, дедуктивная верификация, используются языки программирования, интегрирующие программы и соответствующие

доказательства корректности. Происходит как развитие и рост практического применения традиционных методов автоматической статической верификации, так и появление принципиально новых подходов в этой области. Методы дедуктивной верификации развиваются в направлении интеграции автоматических и «ручных» подходов к доказательству корректности программ, а также использования преимуществ специализированных систем типов и автоматизации проверки свойств программ, работающих с динамическими структурами данных.

Во многих современных инструментах статической верификации используются SMT-решатели, и, таким образом, семантика языка программирования, на котором написана верифицируемая программа, частично или полностью моделируется с помощью логических формул в соответствующих теориях (SMT-формул). В зависимости от используемого языка программирования (например, императивного, объектно-ориентированного или функционального, сильно или слабо, статически или динамически типизированного) и языка спецификации (в частности, основанного на логике первого порядка, высших порядков или, например, логике разделения), а также от используемого метода верификации (автоматическая статическая или дедуктивная верификация), подходы к соответствующему эффективному моделированию семантики в виде SMT-формул могут очень существенно различаться как друг от друга, так и от подходов к заданию (моделированию) семантики тех же языков программирования (например, в соответствующей документации, стандартах, руководствах и т. д.). Даже небольшие изменения в одном методе моделирования семантики некоторых фиксированных языков программирования и спецификации (например, Си и ACSL) могут приводить к изменению, к примеру, времени работы SMT-решателей на результирующих формулах (и, как следствие, времени работы всего инструмента верификации) в несколько десятков раз. Кроме этого, методы моделирования семантики языков программирования различаются не только по производительности SMT-решателей на результирующих формулах, но и по ряду специфических свойств, которые могут быть в различной степени важны при

использовании соответствующего метода в том или ином инструменте статической верификации. В таком контексте разработка соответствующих методов эффективного моделирования семантики используемых языков программирования и спецификации с помощью SMT-формул становится важной и актуальной задачей.

Для языка Си основной проблемой при моделировании семантики в виде логических формул является моделирование семантики операций с указателями, в частности, указателями на динамически выделяемые области памяти наперед не ограниченного размера, а также моделирование различных приведений типов указателей и случаев использования объединений. Это делает актуальной задачу разработки соответствующих методов эффективного и точного моделирования памяти Си-программ для используемых на практике инструментов статической верификации.

Помимо развития собственно методов моделирования памяти Си-программ важной задачей является также оценка эффективности этих методов при использовании для верификации реальных промышленных программных систем.

Таким образом, можно выделить следующие **цель** и **задачи** данной работы.

Цель и задачи работы

Цель работы – разработка и реализация методов моделирования памяти Си-программ, адаптированных для модулей ядра ОС Linux, для применения в инструментах автоматической статической и дедуктивной верификации, использующих SMT-решатели.

Для достижения цели работы были поставлены следующие **задачи**:

- Провести анализ существующих методов моделирования памяти Си-программ в инструментах автоматической статической и дедуктивной верификации.
- Выявить требования к моделям памяти, наиболее подходящим для применения в инструментах автоматической статической и дедуктивной верификации, используемых на практике для модулей ядра ОС Linux.

- Разработать модели памяти для практически используемых инструментов автоматической статической и дедуктивной верификации, отвечающих выявленным требованиям.
- Провести теоретическое обоснование корректности и полноты разработанной модели памяти для инструмента дедуктивной верификации.
- Реализовать предложенные модели памяти в используемых на практике инструментах верификации.
- Провести практическое сравнение эффективности разработанных моделей памяти с ранее реализованными моделями, в том числе для выявления направлений дальнейшего развития.

Научная новизна работы

Научной новизной обладают следующие **результаты работы**:

- Модель памяти на основе теории неинтерпретируемых функций для автоматической статической верификации Си-программ с использованием предикатных абстракций, уточняемых с помощью интерполяции Крейга.
- Полная модель памяти с поддержкой вложенных структур и массивов, а также объединений и переинтерпретации типов указателей с автоматизированным разделением памяти на непересекающиеся области (регионы) для дедуктивной верификации Си-программ.
- Формализация низкоуровневой семантики практически значимого подмножества языка Си, а также соответствующие формальные доказательства корректности и полноты модели памяти с поддержкой вложенных структур и массивов для дедуктивной верификации Си-программ относительно этой семантики.
- Доказательство корректности и полноты модели памяти для дедуктивной верификации Си-программ с разделением на непересекающиеся регионы.
- Метод автоматизированного разделения на непересекающиеся регионы для соответствующей модели памяти, полный (полностью автоматический) для ограниченного класса Си-программ.

- Доказательство полноты метода автоматического разделения на непересекающиеся регионы при использовании соответствующей модели памяти для ограниченного класса Си-программ.

Теоретическая и практическая значимость

Предложено две модели памяти для использования в инструментах автоматической статической и дедуктивной верификации Си-программ.

Реализация модели на основе теории неинтерпретируемых функций в инструменте статической верификации CPAchecker позволила уменьшить число ложных сообщений об ошибках при верификации модулей ядра ОС Linux в рамках проекта LDV, а также верифицировать модули ядра относительно новых правил корректности, существенно использующих адресную арифметику.

Полная модель памяти с поддержкой вложенных структур и массивов, а также объединений и переинтерпретации типов указателей с автоматизированным разделением на непересекающиеся регионы, была разработана и реализована в инструменте дедуктивной верификации Си-программ Jessie. Разработанная модель памяти позволила адаптировать инструмент Jessie для дедуктивной верификации модулей ядра ОС Linux. За счет этого в инструменте Jessie удалось обеспечить поддержку вложенных структур и переинтерпретации типов указателей на целочисленные значения. Предложенная модель памяти также расширяет область применимости реализованной ранее в инструменте Jessie модели памяти с регионами, а также позволяет моделировать семантику Си-программ более полно в сравнении с исходной версией Jessie.

Реализованные механизмы моделирования памяти могут использоваться как в исследовательских, так и в производственных задачах. Они опубликованы под открытой лицензией. Заинтересованными отечественными пользователями этих результатов могут быть такие исследовательские центры как МГУ, СПбГУ, МВТУ, СПбПУ, ФГУ ФНЦ НИИСИ РАН, ИПМ им. М.В. Келдыша РАН, АО «НПО РусБИТех» и др.

Модифицированный инструмент дедуктивной верификации Jessie может быть использован при обучении студентов в рамках курса формальной

верификации программ в таких университетах как МГУ, СПбГУ, НИУ ВШЭ и СПбПУ.

Публикации и зарегистрированные программы

Всего по теме диссертации автором опубликовано 10 работ [1—10]. Основные результаты опубликованы в работах [2,4,6,10]. Работы [1—10] опубликованы в изданиях перечня ВАК, 9 работ ([1—9]) входят в Scopus, работа [6] опубликована в сборнике трудов международной конференции. В работах [2,4] Хорошилов А. В. консультировал автора работы по вопросам возможностей языка Си, используемым в коде модулей ядра ОС Linux, а также по набору видов свойств, проверяемых для модуля безопасности в рамках проекта Astraver. Работа [10] написана совместно с Мутилиным В. С. на основе выполненной автором реализации и краткого формального описания соответствующего метода. В работе [6] автором написан раздел, описывающий предикатный анализ, а также соответствующая реализация построителя формул для этого анализа. В ходе выполнения работы было получено 6 свидетельств о государственной регистрации программ для ЭВМ [1–6], среди которых непосредственно автором работы была выполнена реализация четырех программ ([1–3,6]).

Апробация работы

Основные результаты работы докладывались на следующих конференциях и семинарах:

1. Двадцатаямеждународнаянаучно-техническаяконференция «International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)» (Гренобль, Франция, 5-13 апреля 2014).
2. Международная научно-практическая конференция «Tools&Methods of Program Analysis» (Кострома, Россия, 14-15 ноября 2014 года).
3. Научно-исследовательский семинар Института системного программирования РАН.

4. Международныйсимпозиум «International Symposium On Leveraging Applications of Formal Methods, Verification and Validation» (Amirandes, Гераклион, Крит, Греция, 18-20 октября 2010).
5. Научно-исследовательский семинар лаборатории «Software and Computational Systems Lab» Университета Пассау, Германия.

Личный вклад

Все представленные результаты работы были получены автором лично.

Структура и объем диссертации

Работа состоит из введения, четырех глав, заключения, списка литературы (174 наименования) и одного приложения. Основной текст диссертации (без приложений и списка литературы) занимает 165 страниц.

Краткое содержание работы

Во **введении** представлена краткая характеристика работы, рассматриваются вопросы актуальности работы и научной новизны полученных результатов.

В **первой главе** приведен обзор существующих подходов к моделированию состояния памяти Си-программ в инструментах статической верификации. Представлен обзор основных подходов к статической верификации программ и соответствующих инструментов статической верификации, а также подходов к моделированию состояния памяти программ в этих инструментах, в частности, для языка Си.

В разделе 1.1 представлена краткая характеристика основных подходов к статической верификации программ. Выделено три основных группы подходов:

1. Подходы, рассматривающие программы как совокупности трасс выполнения. В этом подходе проверяемые свойства записываются в виде формул в логиках, семантика которых определяется для трасс выполнения программ (например, темпоральная логика). К этой группе, в частности, относятся классические разновидности проверки моделей программ, а также статического анализа.
2. Дедуктивная верификация программ, в которой программы рассматриваются как наборы функций, представляющих собой композиции

конструкций некоторого языка программирования. Проверяемые свойства формулируются в виде контрактных спецификаций (пред- и постусловий) проверяемых функций. К этой группе относятся инструменты верификации, основанные на исчислении преобразователей предикатов, например, слабейшего предусловия.

3. Подходы, основанные на денотационной семантике, рассматривающие программы как математические объекты, или, иначе говоря, сопоставляющие компонентам программы соответствующие математические сущности – денотации. Проверяемые свойства программ формулируются в виде математических утверждений о соответствующих денотациях, например, в виде теорем в логике высшего порядка.

В разделе 1.2 перечислены основные варианты использования SMT-решателей в различных инструментах статической верификации. Затем выделена общая схема использования SMT-решателей в инструментах верификации (рис. 1).

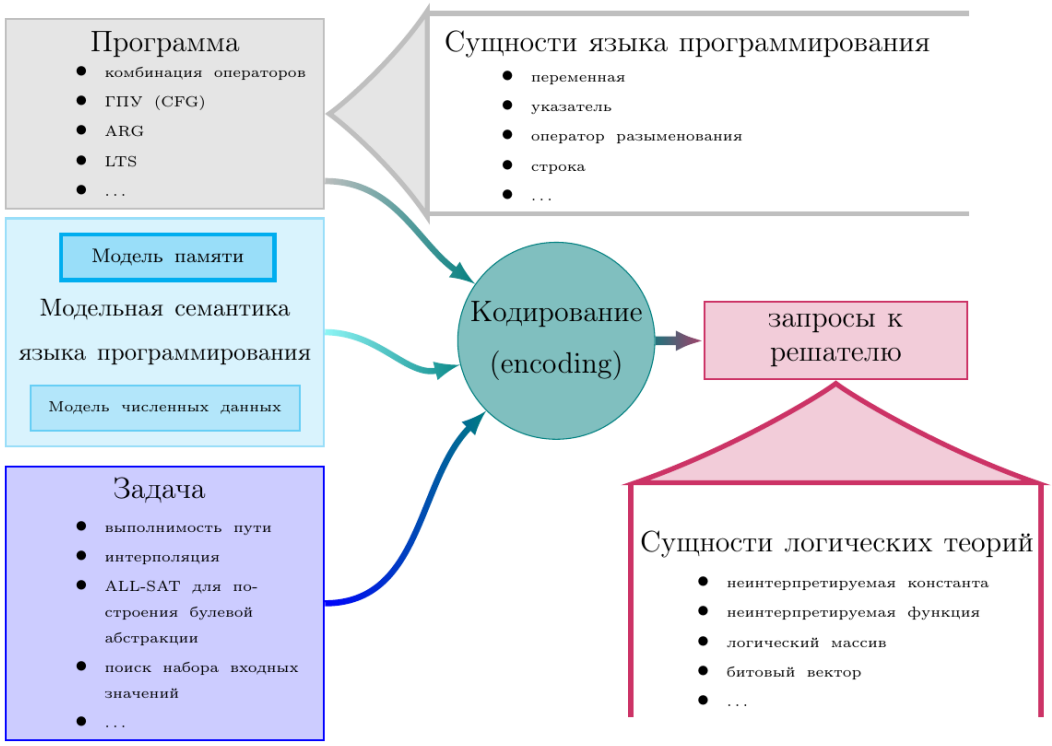

Рисунок 1. Общая схема использования SMT-решателей в инструментах верификации.

Согласно схеме, сущности исходной программы, соответствующие сущностям используемого языка программирования, с использованием **модельной семантики** этого языка и с учетом текущей задачи инструмента верификации преобразуются в сущности поддерживаемых SMT-решателем логических теорий. В зависимости от используемого языка программирования различие между модельной семантикой языка программирования и его изначально заданной (например, в стандарте языка или при рассмотрении метатеории языка, например, его системы типов) семантикой может варьироваться в очень широких пределах.

В разделе 1.3 обосновывается актуальность задачи статической верификации программ на языке Си и, как следствие, актуальность разработки соответствующих модельных семантик этого языка для различных инструментов статической верификации.

В разделе 1.4 указываются логики (комбинации теорий), используемые для моделирования семантики языка Си в методах, предлагаемых далее в работе, а также приводится ссылка на Приложение А, которое содержит краткий обзор основных теорий, поддерживаемых современными SMT-решателями.

В разделе 1.5 обозначены основные проблемы, возникающие при разработке модельных семантик для языка Си. Так как Си является императивным и слабо типизированным языком программирования, основные проблемы моделирования семантики Си-программ в виде логических формул связаны с возможностью произвольной **синонимичности указателей**, то есть произвольных случаев пересечения областей памяти, адресуемых различными указательными выражениями.

В разделе 1.6 приводится обзор существующих методов моделирования семантики операций с указателями в Си-программах. Соответствующие методы названы **моделями памяти**. Рассматриваемые в разделе модели памяти характеризуются со следующих точек зрения:

- Логические теории, используемые при моделировании.

- Возможности языка Си, поддерживаемые моделью памяти. Многие распространенные методы моделирования памяти Си-программ являются в той или иной степени неполными.
- Применимость модели для областей памяти наперед не ограниченного размера.
- Масштабируемость модели памяти. Под масштабируемостью понимается типичная длина простых путей в Си-программе, для которых используется модель.

Среди возможностей языка Си выделяется возможность использования областей памяти наперед не ограниченного размера. В соответствии с поддержкой этой возможности рассмотренные в разделе модели памяти разделены на две группы – модели для ограниченных областей памяти и модели памяти неограниченного размера. Среди моделей для ограниченных областей памяти рассматриваются следующие:

- Модель, использующая результаты предварительного анализа алиасов. Эта модель совместима с использованием теорий линейной целочисленной или вещественной арифметики, а также теории битовых векторов конечной длины. Модель не поддерживает массивы, адресную арифметику, объединения и приведения типов указателей. Применяется к простым путям длиной до нескольких тысяч операторов, включая объявления.
- Модель на основе исчисления слабейших предусловий, использующая теорию неинтерпретируемых функций. Эта модель предполагает использование логики первого порядка и теории неинтерпретируемых функций, а также теории целочисленной или вещественной линейной арифметики. Модель позволяет поддерживать массивы и адресную арифметику, но не объединения или приведения типов указателей и обладает примерно такой же масштабируемостью, что и модель, использующая результаты анализа алиасов.

Среди моделей для памяти неограниченного размера рассматриваются:

- Три модели памяти, предполагающие использование либо теории массивов, либо логики первого порядка и теории неинтерпретируемых функций, а также теории целочисленной или вещественной линейной арифметики или теории битовых векторов конечной длины: нетипизированная модель, типизированная модель и модель Бурсталла-Борната. Все модели полностью поддерживают все возможности языка Си и рассчитаны на применение к коротким базовым путям длиной до нескольких десятков операторов. Модели различаются производительностью в смысле ожидаемого среднего времени работы SMT-решателей на результирующих формулах.
- Модель памяти с регионами, использующая те же теории, что и три предыдущие модели, однако обладающая большей производительностью и меньшей полнотой, а именно неполнотой поддержки вложенных структур и массивов, а также произвольных объединений и приведений типов указателей.
- Модель памяти, используемая в инструменте GRASShopper, основанная на использовании фрагмента сепарационной логики GRASS и таким образом поддерживающая динамические структуры данных со списочным скелетом (в частности, одно- и двусвязные списки и бинарные деревья поиска). Модель предполагает использование алгоритмически разрешимого фрагмента комбинации логики первого порядка и теорий неинтерпретируемых функций и линейной целочисленной арифметики – формул, стратифицированных по сортам.

Масштабируемость всех рассмотренных моделей памяти для неограниченных областей памяти примерно одинакова.

Далее проводится сравнение всех рассмотренных в разделе моделей памяти на простом примере специфицированной на языке ACSL Си-функции. Сравнение показывает значительные различия в производительности как SMT-решателей, так и инструментов верификации в целом в зависимости от используемой при построении логических формул модели памяти.

Глава завершается обобщением результатов исследований впо сравнению производительности SMT-решателей при использовании различных моделей памяти и делается вывод о существенном влиянии используемой модели памяти как на область применимости соответствующего инструмента верификации, так и на его производительность.

Во **второй главе** кратко рассмотрены два проекта, активно использующих инструменты статической верификации с целью выявления технических и архитектурных ограничений, в рамках которых нужно искать решение по развитию механизмов реализации моделей памяти.

В первом проекте – LDV по статической верификации модулей ядра ОС Linux используются инструменты автоматической статической верификации и, в частности, инструменты, основанные на применении предикатной абстракции с уточнением, предполагающим использование интерполяции Крейга для невыполнимых логических формул, соответствующих путям выполнения программы. В контексте этого проекта актуальной является проблема поддержки инструментами верификации таких распространенных в модулях ядра возможностей языка Си как динамическое выделение памяти, массивы и адресная арифметика. В силу отсутствия такой поддержки в используемых в проекте LDV на момент начала выполнения данной работы инструментах автоматической статической верификации, появилась задача разработки соответствующей модели памяти, поддерживающей динамическую память, массивы и адресную арифметику и совместимой с применением интерполяции Крейга.

Во втором проекте – Astraver по дедуктивной верификации модулей ядра ОС Linux используются инструменты дедуктивной верификации, совместимые с платформой статического анализа Frama-C, а именно модули WP и Jessie. Модуль Jessie реализует более эффективную модель памяти с регионами, которая, однако, обладает неполнотой для вложенных структур и массивов, а также произвольных объединений и приведений типов указателей. Кроме этого, в силу изначальной адаптации для промежуточного языка Jessie, для модели памяти с регионами не была доказана корректность соответствующей модельной семантики

относительно некоторой низкоуровневой семантики языка Си. Соответственно в контексте проекта Astraver появляется задача разработки корректной и полной модели памяти с регионами для инструмента дедуктивной верификации Jessie.

Выводом проведенного рассмотрения явилась постановка задачи разработки моделей памяти для двух инструментов статической верификации: инструмента автоматической статической верификации CPAchecker и инструмента дедуктивной верификации Jessie. Разрабатываемые модели памяти должны удовлетворять различным требованиям областей применения соответствующих инструментов верификации. Для модели памяти в инструменте CPAchecker требуется максимальная эффективность и совместимость с интерполяцией Крейга при возможной потере полноты и даже корректности в общем случае. Для модели памяти в инструменте Jessie требуется формально доказанная полнота и корректность модели памяти, но допустимо использование произвольных поддерживаемых решателями логических теорий.

В **третьей главе** представлен предлагаемый в данной работе метод моделирования состояния памяти Си-программы для инструмента автоматической статической верификации, использующего предикатные абстракции, уточняемые с помощью интерполяции Крейга. В методе при построении формул используется комбинация теории неинтерпретируемых функций с теорией линейной целочисленной или вещественно арифметики. Метод поддерживает динамическое выделение памяти ограниченного, то есть явно известного и фиксированного для каждого рассматриваемого пути выполнения, размера, а также массивы конечной длины и адресную арифметику. В главе дан как обзор метода, так и его формальное описание в рамках концепции конфигурируемого анализа (Configurable Program Analysis, CPA).

В разделе 3.1 приведен обзор метода и перечислены основные идеи по представлению состояния памяти, операции обновления значения по указателю и условий непересечения адресов выделенных в памяти объектов с помощью неинтерпретируемых функций. Также для предлагаемого метода обоснована выполненность основного требования полноты – надежного уточнения

абстракции, необходимого для методов моделирования памяти, предполагающих использование интерполяции Крейга для уточнения абстракции.

В разделе 3.2 представлено формальное описание предлагаемого метода. В подразделе 3.2.1 даны определения автомата потока управления, конкретного состояния программы, пути в программе, формулы пути, достижимости ошибочного состояния, точности для предикатной абстракции и соответствующего абстрактного состояния данных, а также кодирования с настраиваемым размером блока и соответствующие определения состояния абстракции, состояния без абстракции и дизъюнктивной формулы пути. В подразделах 3.2.2 и 3.2.3 предлагаемая модель памяти описана с помощью правил построения логических ограничений, составляющих формулу пути, и изменения абстрактного состояния данных в соответствии с операциями на дугах автомата потока управления. В подразделе 3.2.4 построение формулы пути рассмотрено на небольшом примере. В подразделе 3.2.5 дается описание конфигурируемого анализа (CPA), соответствующего предикатной абстракции с использованием предлагаемого метода, и формальные определения всех его составляющих компонент – абстрактного домена, отношения перехода, оператора слияния и оператора останова.

В разделе 3.3 описаны оптимизации предложенной модели памяти, реализованные в инструменте верификации CPAchecker.

В разделе 3.4 приведены результаты сравнения версий инструмента CPAchecker до и после реализации в нем предложенной модели памяти на тестовых наборах SV-COMP. Также приведены результаты сравнения с конфигурациями последующих версий инструмента CPAchecker, в которых разработчиками инструмента были также реализованы две другие модели памяти – на основе логики первого порядка и неинтерпретируемых функций и на основе теории массивов.

В таблице 1 приведены результаты сравнения основных используемых версий инструмента CPAchecker до и после реализации в нем предложенной модели

памяти на тестовом наборе DeviceDrivers64, состоящем из задач верификации, полученных для модулей ядра ОС Linux с помощью инструментария LDV. Столбец «без НФ» соответствует версии инструмента до интеграции в него предложенной модели памяти. В соответствующей конфигурации при анализе программ не моделируются динамические выделения памяти, обращения к элементам массивов и адресная арифметика, вместо этого используется приближение недетерминированными значениями. Столбец «с НФ» соответствует версии инструмента с интегрированной в него предложенной моделью памяти, использующей теорию неинтерпретируемых функций.

Модели памяти	без НФ	с НФ
Общее количество	2121	2121
Корректные результаты за 900с	1654	1643
Доказано отсутствие ошибки	1450	1450
Ошибка найдена	204	193
Некорректные результаты за 900с	17	6
Упущенная ошибка	5	3
Ложное предупреждение	12	3
Завершено по истечении 900 с	450	472
Общее время	140 часов	144 часа
Время для корректных результатов	22,2 часов	22,7 часов

Таблица 1. Результаты сравнения версий инструмента CPAchecker в основной конфигурации, используемой в проекте LDV, на наборе DeviceDrivers64.

Результаты сравнения версий инструмента CPAchecker показывают снижение числа некорректных результатов верификации.

В разделе 3.5 подводятся итоги разработки и реализации модели памяти с использованием теории неинтерпретируемых функций для инструмента автоматической статической верификации CPAchecker. Модель памяти позволила увеличить точность предикатного анализа в инструменте без значительных потерь в его эффективности. Реализация соответствующего предложенной модели памяти алгоритма построения формул пути в инструменте CPAchecker позволяет повторно использовать ее для различных видов анализа программ, отличных от предикатной абстракции с уточнением. Из поддерживаемых инструментом видов статического анализа соответствующая реализация на момент написания работы использовалась по крайней мере еще в трех.

В **четвертой главе** представлена предлагаемая в работе модель памяти с вложенными регионами для дедуктивной верификации Си-программ. Модель предполагает использование логики первого порядка с теорией неинтерпретируемых функций и теорией линейной или нелинейной целочисленной арифметики, либо теорией битовых векторов конечной длины. Вместо теории неинтерпретируемых функций возможно использование теории массивов. Модель памяти поддерживает все возможности языка Си, для неё формально показана корректность модельной семантики относительно низкоуровневой семантики языка Си. Рассмотрение предлагаемой модели памяти проводится в несколько этапов.

В разделе 4.1 вводится базовый язык с поддержкой вложенных структур и массивов. Базовый язык содержит только конструкции, необходимые для представления базовых путей, то есть не является процедурным и не содержит операторов циклов и переходов. Для языка вводится система типов с указателями, параметризованными *регионами*, объясняется неформальный смысл вводимых в базовый языка спецификационных конструкций.

В подразделе 4.1.2 для базового языка с поддержкой вложенности формально задается исходная (низкоуровневая) семантика, аналогичная семантике

соответствующего подмножества языка Си. Исходная семантика включает в себя спецификационные конструкции, относительно которых проверяется корректность задаваемых на базовом языке базовых путей. Для путей на базовом языке проверяется в том числе корректность работы с памятью, для чего в базовый язык включены соответствующие операции выделения и освобождения памяти. Семантика задается с помощью правил вывода для недетерминированного отношения вычисления, которое связывает базовый путь с соответствующим подмножеством значений из множества$\{\top, \bot\}$, где $\top$ соответствует успешному выполнению базового пути, а $\bot$ – нарушению одного из условий верификации.

В разделе 4.2 описываются преобразования исходных Си-программ, необходимые для представления их базовых путей на базовом языке с поддержкой вложенности.

В разделе 4.3 формально задается модельная семантика базового языка с помощью соответствующего отношения вычисления, аналогичного отношению вычисления, используемому для формализации исходной семантики. Основное отличие модельной семантики состоит в использовании отдельных логических сущностей – массивов или неинтерпретируемых функций для моделирования состояния различных *регионов* – областей памяти, для соответствующих множеств адресов. Регионы, используемые в модельной семантике соответствуют регионам из указательных типов в путях на базовом языке.

В подразделе 4.3.1 вводится инвариант, связывающий состояния памяти программы при выполнении базового пути в исходной и модельной семантике. С помощью введенного инварианта доказывается теорема о корректности модельной семантики базового языка относительно его исходной семантики.

В подразделе 4.3.2 доказывается теорема о полноте модельной семантики в смысле существования преобразования исходного базового пути, сохраняющего его исходную семантику, такого, что при условии равенства значения базового пути значению $\{\top\}$ в исходной семантике (корректность всех проверяемых свойств для любого возможного выполнения), значение преобразованного базового пути в модельной семантике также равно $\{\top\}$. В теореме о полноте

модельной семантики также делаются дополнительные предположения о типизации указателей путей на базовом языке

В разделе 4.4 рассматривается подход к автоматическому выводу регионов для указательных типов в путях на базовом языке. Подход описывается неявно в виде дополнительных ограничений на значения регионов указателей в контексте типизации путей на базовом языке. Вводятся дополнительные ограничения на исходные Си-программы, для которых гарантируется полнота модельной семантики при выполнении введенных дополнительных ограничений на контекст типизации. Приводится план доказательства соответствующей теоремы о полноте.

В разделе 4.5 подводятся итоги разработки модели памяти с вложенными регионами для дедуктивной верификации Си-программ. Разработанная модели памяти позволяет расширить область применимости инструмента дедуктивной верификации Jessie на Си-программы, существенно использующие вложенные структуры и массивы, а также объединения и произвольные приведения типов указателей, без потери эффективности ранее реализованной в инструменте модели памяти с регионами. Разработанная модель памяти также предполагает более простой способ моделирования объединений и приведений типов указателей, не требующий введения с этой целью специального модельного состояния (используемой в исходном инструменте Jessie модельной таблицы тегов объектов). Моделирование этих возможностей языка Си интегрировано с моделированием защиты памяти и использует соответствующее модельное состояние (таблицы аллокации). Для разработанной модели памяти доказана корректность и полнота относительно практически значимого фрагмента низкоуровневой семантики языка Си, включающего поддержку защищенной динамической памяти, вложенных структур и массивов, а также объединений и приведений типов указателей. Предлагаемая модель памяти может быть достаточно легко расширена для поддержки структур и объединений с произвольным выравниванием полей, а также битовых полей структур и объединений. Частичная реализация предложенной модели памяти в инструменте Jessie – реализация поддержки вложенных структур и массивов, а также приведения типов указателей на

целочисленные типы данных – позволила применить инструмент для дедуктивной верификации модуля безопасности ядра ОС Linux.

В заключении представлены основные научные и практические **результаты работы**, выносимые на защиту:

- Разработана модель памяти на основе теории неинтерпретируемых функций для автоматической статической верификации Си-программ с использованием предикатных абстракций, уточняемых с помощью интерполяции Крейга. Разработанная модель памяти на основе теории неинтерпретируемых функций интегрирована в инструмент автоматической статической верификации CPAchecker.
- Разработана полная модель памяти с поддержкой вложенных структур и массивов, а также объединений и переинтерпретации типов указателей с автоматизированнымразделенем на непересекающиеся области (регионы) для дедуктивной верификации Си-программ.
- Предложена формализация низкоуровневой семантики практически значимого подмножества языка Си, включающего поддержку защищенной динамической памяти, вложенных структур и массивов, а также объединений и приведений типов указателей, а также соответствующие формальные доказательства корректности и полноты модели памяти с поддержкой вложенных структур и массивов для дедуктивной верификации Си-программ относительно этой семантики.
- Предложена схема доказательства полноты метода автоматического разделения на непересекающиеся регионы при использовании соответствующей модели памяти для ограниченного класса Си-программ.
- В инструменте дедуктивной верификации Jessie была реализована поддержка переинтерпретации типов указателей на целочисленные типы данных.

В **Приложении А** приведен краткий обзор основных теорий, поддерживаемых современными SMT-решателями. Для каждой рассматриваемой теории описана грамматика формул, аксиоматизация в случаях, когда она не является широко известной, а также алгоритмическая разрешимость и/или сложность алгоритмов решения задачи о выполнимости формул в соответствующей теории. Кратко рассмотрены также наиболее широко применяемые на практике комбинации логических теорий.

Список работ, опубликованных автором по теме диссертации

1. Mandrykin, M. U. Towards deductive verification of C programs with shared data / M. U. Mandrykin, A. V. Khoroshilov // Programming and Computer Software. — 2016. — Vol. 42. — no. 5. — P. 324-332.
2. Mandrykin, M. U. Region analysis for deductive verification of C programs / M. U. Mandrykin, A. V. Khoroshilov // Programming and Computer Software. — 2016. — Vol. 42. — no. 5. — P. 257–278.
3. Mandrykin, M. Towards deductive verification of concurrent Linux kernel code with Jessie / M. Mandrykin, A. Khoroshilov // Computer Science and Information Technologies (CSIT) 2015. — 2015. — P. 5-10.
4. Mandrykin, M. U. High-level memory model with low-level pointer cast support for Jessie intermediate language / M. U. Mandrykin, A. V. Khoroshilov // Programming and Computer Software. — 2015. — Vol. 41. — no. 4. — P. 197–207.
5. Zakharov, I. S. Configurable toolset for static verification of operating systems kernel modules / I. S. Zakharov, M. U. Mandrykin, V. S. Mutilin, E. M. Novikov, A. K. Petrenko, A. V. Khoroshilov // Programming and Computer Software. — 2015. — Vol. 41. — no. 1. — P. 49–64.
6. Löwe, S. CPAchecker with Sequential Combination of Explicit-Value Analyses and Predicate Analyses / S. Löwe, M. Mandrykin, P. Wendler // Proceedings of Tools and Algorithms for the Construction and Analysis of Systems: 20th International Conference — 2014. — P. 392-394.
7. Mandrykin, M. U. Using Linux Device Drivers for Static Verification Tools Benchmarking / M. U. Mandrykin, V. S. Mutilin, E. M. Novikov et al. // Program. Comput. Softw. — 2012. — Vol. 38. — no. 5. — P. 245–256.
8. Shved, P. E. Experience of Improving the Blast Static Verification Tool / P.Shved, V. S. Mutilin, M. U. Mandrykin // Program. Comput. Softw. — 2012. — Vol. 38. — no. 3. — P. 134–142.
9. Shved, P. Predicate Analysis with Blast 2.7 / P.Shved, V. S. Mutilin, M. U. Mandrykin // LNCS, Proceedings of TACAS. — 2012. — Vol. 7214. — P. 525–527.

10. Мандрыкин, М. У. Моделирование памяти с использованием неинтерпретируемых функций в предикатных абстракциях / М. У. Мандрыкин, В. С. Мутилин // Труды Института системного программирования РАН. — 2015. — Vol. 27. — P. 117–142.

Зарегистрированные программы ЭВМ

1. Мандрыкин М.У., Хорошилов А.В. «Программа дедуктивной верификации программ на языке программирования Си, использующая модель памяти с чувствительным к контексту разделением на непересекающиеся регионы». Свидетельство о государственной регистрации программы для ЭВМ № 2016618347 от 27.07.2016.
2. Мандрыкин М.У., Хорошилов А.В. «Система дедуктивной верификации программ с возможностью адаптивного моделирования операций над целочисленными данными». Свидетельство о государственной регистрации программы для ЭВМ № 2016618348 от 27.07.2016.
3. Мандрыкин М.У., Хорошилов А.В. «Программа дедуктивной верификации программ на языке Си с возможностью интерпретации участков памяти как объектов разных типов». Свидетельство о государственной регистрации программы для ЭВМ № 2015617941 от 27.07.2015.
4. Мандрыкин М.У., Мутилин В.С., Хорошилов А.В. «Построитель формул с моделированием памяти для уточнения предикатной абстракции с помощью интерполяции». Свидетельство о государственной регистрации программы для ЭВМ № 2013614375 от 06.05.2013.
5. Швед П.Е., Новиков Е.М., Мандрыкин М.У., Мутилин В.С., Хорошилов А.В. «Система проверки выполнения проблемно-ориентированных правил для Си программ». Свидетельство о государственной регистрации программы для ЭВМ № 2012615596 от 20.06.2012.
6. Мандрыкин М.У., Мутилин В.С., Хорошилов А.В. «Интерполирующий решатель, поддерживающий формулы с кванторами в теории линейной арифметики и неинтерпретируемых функций». Свидетельство о государственной регистрации программы для ЭВМ № 2012618566 от 21.09.2012.