На правах рукописи

Герасимов Александр Юрьевич

Классификация предупреждений о программных ошибках методом динамического символьного исполнения программ

Специальность 05.13.11 –
«Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей»

Автореферат
диссертации на соискание учёной 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: Кандидат физико-математических наук **Курмангалеев Шамиль Фаимович**

Официальные оппоненты: **Ильин Вячеслав Анатольевич**
доктор физико-математических наук
старший научный сотрудник,
начальник отдела Курчатовского комплекса НБИКС-природоподобных технологий Национального исследовательского центра «Курчатовский институт»

Волконский Владимир Юрьевич
кандидат технических наук,
начальник отделения
«Системы программирования» Публичного акционерного общества «Институт электронных управляющих машин им. И. С. Брука»

Ведущая организация: Федеральное государственное учреждение «Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук»

Защита состоится 14 марта 2019 года в 16:00 на заседании диссертационного совета Д 002.087.01 при Федеральном государственном бюджетном учреждении науки «Институт системного программирования им. В.П. Иванникова Российской академии наук» 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 «____» _____ 201__ года.

Учёный секретарь
Диссертационного совета Д 002.087.01,
Кандидат физико-математических наук Зеленов С.В.

Общая характеристика работы

Актуальность темы исследования

Размер и сложность программного обеспечения (ПО) постоянно увеличиваются. Размер исходного текста современных программ и программных систем может достигать сотен миллионов строк. Усложнение приводит к тому, что создание качественного программного обеспечения становится фактически невозможным без применения автоматических средств проверки программ на соответствие функциональным требованиям к программному обеспечению и на отсутствие программных ошибок. Наличие ошибок в программном обеспечении в первую очередь связано с тем, что совершенствование подготовки программистов не успевает за увеличением сложности и размера программ. Несмотря на развитие методов разработки программного обеспечения и инструментальных средств поддержки процесса разработки программ, в выпускаемых программах содержатся ошибки, которые могут приводить к неправильной работе программы, несанкционированному доступу к критическим данным и выполнению злонамеренного кода. По данным портала CVE Details (www.cvedetails.com), предоставляющего статистику зарегистрированных ошибок в программах на основе информации корпорации MITRE с 1999 года, ежегодно регистрируется несколько тысяч критических ошибок в используемых программах.

С начала 2000-х годов наблюдается значительный рост индустрии разработки средств автоматического обнаружения ошибок в программах. Среди них выделяют методы статического и динамического анализа программ. Методы статического анализа программ позволяют обнаруживать ошибки в программном обеспечении без запуска программ на исполнение. Динамические методы анализа программ позволяют обнаружить ошибки в программах в процессе (online) или по результатам (offline, postmortem) исполнения программы. У каждого из методов обнаружения ошибок в программах есть как преимущества, так и недостатки.

Методы статического анализа программ позволяют проанализировать программу целиком, включая части программ, исполняющиеся крайне редко, обладают высокой производительностью анализа, производя полный анализ программ

размером в несколько миллионов строк исходного текста, хорошо масштабируются. Но при этом выявление свойств программы методами статического анализа является в общем случае алгоритмически неразрешимой задачей. В связи с этим в процессе создания инструментальных средств статического анализа приходится идти на компромиссы между количеством истинных предупреждений об ошибках и количеством необнаруженных ошибок для удовлетворения требований масштабируемости и производительности анализа, что приводит к формированию результата анализа программ в виде гипотез о наличии ошибок в программе.

Оценка качества инструментов, предназначенных для обнаружения программных ошибок, производится путем определения соотношения *количества истинных предупреждений* об ошибке (принятая гипотеза о наличии ошибки верна), *количества ложных предупреждений* об ошибке (принятая гипотеза о наличии ошибки неверна: ложноположительное предупреждение, или ошибка первого рода) и *количества ненайденных ошибок* (принятая гипотеза об отсутствии ошибки неверна: ложноотрицательный результат анализа, или ошибка второго рода). Точность анализа выражается соотношением количества истинных предупреждений об ошибках и общим количеством предупреждений об ошибках в программе. Уровень ошибок первого рода при анализе таких программных систем, как ОС Android, может достигать 40% от общего количества предупреждений об ошибках, что в абсолютных величинах составляет тысячи предупреждений. Квалифицированному специалисту на анализ одного предупреждения об ошибке может потребоваться значительное время, что приводит к необоснованным временным затратам и нежелательным финансовым расходам, в связи с большой долей ошибок первого рода в результатах анализатора. Возникает задача автоматической классификации предупреждений об ошибках в программе с целью предварительного разбиения их на классы: заведомо истинные предупреждения, заведомо ложные предупреждения и предупреждения, истинность или ложность которых в автоматическом режиме гарантированно установить не удается.

Для решения задачи классификации программных ошибок, найденных методами статического анализа программ, предлагается применять методы динамического анализа программ. Методы

динамического анализа программ позволяют автоматически обнаруживать ошибки в программах и слабо подвержены влиянию проблемы ошибок первого рода, так как позволяют вычислять наборы внешних (входных) данных программы, на которых можно продемонстрировать ошибку в программе и использовать их для отладки программы.

В данной работе предлагается использовать метод динамического символьного исполнения программ для вычисления внешних данных программы, который заключается в представлении потока данных программы в виде набора символьных (свободных) переменных, значения которых зависят от внешних данных программы и операций над ними. Этот метод позволяет в процессе анализа потока данных программы вычислять значения символьных переменных и генерировать новые внешние данные для исполнения программы по альтернативному пути или для проявления ошибки при исполнении потенциально опасных операций.

Несмотря на отсутствие ошибок первого рода при применении метода динамического символьного исполнения программ для решения задачи вычисления наборов внешних данных программы, данный метод отличается низкой производительностью в связи с тем, что для его реализации требуется «тяжеловесная» инструментация программы с целью сбора трассы исполнения программы и решение задачи определения вычислимости формул в теориях для генерации внешних данных программы. Недостатком данного метода также является экспоненциальный рост количества путей в программе от количества условных переходов, зависящих от потока символьных данных программы. Поэтому исчерпывающий анализ программ за ограниченное время методами динамического символьного исполнения затруднителен.

В связи с вышеизложенным становится актуальной задача комбинирования методов статического и динамического анализа программ для классификации предупреждений об ошибках в программах, найденных методами статического анализа программ.

Цель и задачи диссертационной работы. Цель диссертационной работы – повысить точность анализа программ путем комбинирования методов статического анализа программ и динамического символьного исполнения программ с целью классификации предупреждений об ошибках в программах.

Для достижения поставленной цели сформулированы и решены следующие задачи:

1. Разработка алгоритма комбинированного анализа, объединяющего статический анализ и динамическое символьное исполнение программ.
2. Разработка алгоритма направленного динамического символьного исполнения программ, использующего предупреждение об ошибке, полученное методами статического анализа программ, для вычисления внешних данных программы, приводящих к проявлению ошибки в процессе исполнения программы.
3. Разработка метода классификации предупреждений о программных ошибках при помощи разработанных модели и алгоритмов.

Научная новизна. В работе получены следующие результаты, обладающие научной новизной:

1. Разработана модель обнаружения ошибок в программе в процессе символьного исполнения программы.
2. Разработаны алгоритмы комбинированного анализа программ на основе направленного динамического символьного исполнения программ, использующего предупреждение об ошибке, найденной методами статического анализа исходного кода программ, для вычисления внешних данных программы, приводящих к проявлению программной ошибки в процессе исполнения программы.
3. Показана применимость разработанных алгоритмов для решения задачи классификации предупреждений о программных ошибках, найденных методом статического анализа программ.

Теоретическая и практическая значимость. Разработаны алгоритмы направленного динамического символьного исполнения программы. Предложенные алгоритмы реализованы на основе инструментов статического анализа исходного кода программ Svace и инструмента динамического анализа программ Anxiety, разрабатываемых в Институте системного программирования им. В.П. Иванникова РАН. Показана применимость предложенных алгоритмов для решения задачи классификации предупреждений о программных ошибках, найденных методами статического анализа

программ. Результаты исследования, изложенные в диссертации, могут быть использованы для дальнейших исследований в области анализа программ, создания учебных курсов, а также при создании промышленных анализаторов программ.

Методология и методы исследования. Результаты диссертации были получены с использованием методов и моделей, применяемых при проведении динамического символьного исполнения программ. Математическую основу данной работы составляют теория графов, теория множеств, теория алгоритмов, математическая логика.

Положения, выносимые на защиту:

- модель обнаружения ошибок в программе в процессе символьного исполнения программы;
- алгоритмы комбинированного анализа программ на основе направленного динамического символьного исполнения программ, использующего предупреждение об ошибке, полученное методами статического анализа исходного кода программ, для вычисления внешних данных программы, приводящих к проявлению программной ошибки в процессе исполнения программы;
- метод классификации предупреждений об ошибках в программе, основанный на применении разработанных модели и алгоритмов комбинированного анализа программ.

Апробация результатов. Основные результаты диссертационного докладывались на следующих конференциях:

1. Открытая конференция ИСП РАН им. В.П. Иванникова. (Россия, Москва, 2016).
2. 11th International Conference on Computer Science and Information Technologies (Armenia, Yerevan, 2017)
3. Открытая конференция ИСП РАН им. В.П. Иванникова. (Россия, Москва, 2017).
4. 60-я научная конференция МФТИ (Россия, Долгопрудный, 2017)
5. Ivannikov Memorial Workshop. (Armenia, Yerevan, 2018)

Публикации. Материалы диссертации опубликованы в 9 печатных работах, из них 6 опубликованы в изданиях, входящих в перечень рецензируемых научных изданий ВАК при Минобрнауки РФ [1-3, 6, 8-9], 4 статьи опубликованы в сборниках трудов

конференций [4-7]. Пять из девяти статей [4, 6-9] опубликованы в изданиях, индексируемых в Scopus. В совместных работах [1, 3-4, 9] личный вклад автора заключается в описании метода определения достижимости программных дефектов. В работе [6] личный вклад автора заключается в написании введения и описании реализации инструмента Anxiety. Получены 5 свидетельств о государственной регистрации программы для ЭВМ.

Личный вклад автора. Все представленные в диссертации результаты получены автором лично.

Структура и объем диссертации. Диссертация состоит из введения, 4 глав, заключения и библиографии. Общий объем диссертации 129 страниц, из них 129 страниц текста. Библиография включает 177 наименований.

Краткое содержание работы

Во введении обосновывается актуальность темы исследования, научная новизна исследования, практическая значимость результатов, сформулирована цель работы, представлены научные положения, выносимые на защиту.

В первой главе дается понятие программной ошибки и производится обзор известных методов обнаружения ошибок. Рассматриваются определения ошибки в программном обеспечении по ГОСТ 56939-2014, стандартной классификации программных аномалий IEEE, определение понятий программных аномалий NIST. Указывается на отсутствие единого описания термина программной ошибки в регламентирующих документах и научных работах. Приводится статистика зарегистрированных ошибок за более чем 15 лет и проводится обзор методов обнаружения ошибок в программах. Описываются известные классификации ошибок MITRE CWE, OWASP, SPK. Предлагается собственная классификация программных ошибок:

- Ошибки, приводящие к изменению пользовательских данных;
- Ошибки, приводящие к неавторизованному доступу к пользовательским данным;
- Ошибки, приводящие к исчерпанию системных ресурсов;

- Ошибки, приводящие к аварийному завершению работы программы;
- Ошибки, приводящие к исполнению злонамеренного кода.

Проводится обзор причин появления ошибок в программном коде: совершенствование подготовки программистов не успевает за увеличением сложности и размера программ, недостаточное тестирование программного обеспечения, влияние изменений, вносимых в среду исполнения программы. Делается вывод, что появление программных ошибок в сложных программных системах возможно и необходимо применение методов автоматического анализа программ для их обнаружения.

Проводится обзор методов обнаружения ошибок в программах. Методы обнаружения ошибок в программах разделены на два основных метода: методы статического обнаружения ошибок в программах, основанные на автоматической инспекции кода программы, методы динамического обнаружения ошибок в программах, основанные на исследовании программы в процессе или по результатам её исполнения. Приводится ретроспектива методов анализа программ, начиная с простейших синтаксических анализаторов (*lint, Blast, Magic, RATS, FlowFinder, ITS4, Splint, Uno*), анализаторов промежуточного представления программы (*FileRace, Boon, Archer, CQual, Saturn*), и заканчивая современными методами анализа программ на основе совмещения анализа потока программы и символьного исполнения (*Coverity, Klocwork, Svace*). Приводится классификация методов статического анализа программ по способу анализа (автоматический или полуавтоматический), по используемым языкам программирования для анализируемых программ, типу обнаруживаемых ошибок, по сложности и глубине анализа.

Приводится обзор динамических методов обнаружения ошибок в программах. Среди них выделяются методы псевдослучайной генерации входных данных (фаззинг) и динамическое символьное исполнение. Дается ретроспектива применения методов псевдослучайной генерации входных данных с целью обнаружения ошибок (исследования Миллера, методы SPIKE и PROTOS, инструменты AFL и Fuzzgrind). Даётся ретроспектива методов динамического символьного исполнения программ, изначально применявшихся для генерации тестового покрытия, и развитых в итоге до автоматического обнаружения ошибок в программах.

Рассматриваются инструменты DART, CUTE, EXE, SAGE, S^2E. Приводится классификация методов динамического символьного исполнения.

Подробно рассматриваются преимущества и недостатки методов обнаружения ошибок в программах. Для статических методов анализа программ выделяются характеристики полноты, точности и производительности анализа, связанные обратной зависимостью. Рассматриваются причины, влияющие на качество анализа. Показывается необходимость ограничения количества путей для анализа, ограничения множества значений абстрактных доменов для переменных и необходимость работы в условиях недоступности исходного кода библиотек функций. Обосновывается необходимость компромисса между этими характеристиками при реализации методов. При этом указывается явное преимущество методов, связанное с доступностью всего исходного кода программ, даже того, который исполняется при крайне маловероятных условиях, что позволяет найти ошибки, которые крайне сложно найти другими методами анализа программ. Для методов фаззинга в качестве недостатка указывается случайный характер обнаружения ошибок в программах, при том, что по утверждению разработчиков методов показана высокая эффективность по количеству обнаруженных ошибок в программах. Для методов динамического символьного исполнения в качестве основных проблем показано ограничение области их применения в связи с проблемой экспоненциального роста количества путей для анализа (*от англ.* – path explosion problem), сложности анализа функций с интенсивными математическими вычислениями и проблемы недопомеченности (*от англ.* – underteintedness) или перепомеченности (*от англ.* – overtaintedness) потока данных программы, которые влияют на точность и производительность методов динамического символьного исполнения.

Приведен обзор существующих подходов совмещения методов обнаружения ошибок в программах с целью повышения точности результатов и производительности анализа по сравнению с применением их по отдельности. Показаны преимущества методов, совмещающих статический анализ кода программ и динамического символьного исполнения для повышения производительности анализа путем сокращения путей исполнения программы через отбрасывание путей, не приводящих в точку потенциальной ошибки,

применения эвристики минимального расстояния для достижения определенной точки в программе.

В конце главы делается вывод о том, что совмещение различных методов анализа программ является наиболее перспективным направлением для развития методов обнаружения ошибок в программах.

Во второй главе описываются методы, на которых основывается предложенный в работе метод.

Рассматриваются различные уровни статического анализа исходного кода программ: анализа абстрактного синтаксического дерева, анализ потока программы, анализ чувствительный к путям и межпроцедурный анализ. Приводится обобщенная модель представления результатов работы статического анализа программы: предупреждения об ошибке в виде тройки <*i*, *E*, *s*>, где: ***i*** – точка инициализации ошибочной ситуации в программе, выполнение которой приводит к созданию предусловия для появления ошибки; ***E*** – последовательность событий на трассе исполнения программы в виде операций, которые в конечном итоге приводят к появлению ошибочной ситуации; ***s*** – точка проявления ошибки, то есть операция, исполнение которой после прохождения точек инициализации и событий на трассе приведет к проявлению ошибки.

Приводится алгоритм динамического символьного исполнения программ, который состоит из этапов сбора трассы исполнения программы; преобразования трассы в систему символьных формул, в которых входные данные программы представлены в виде свободных переменных (символов), а поток программы – в виде формул, описывающих операции над входными и внутренними данными программы; этапа разбиения трассы программы для вычисления входных данных, приводящих к исполнению по альтернативному пути или к проявлению ошибки в программе; этапа вычисления нового набора входных данных для проведения анализа по альтернативному пути или подтверждения наличия ошибки в программе. Подробно рассматриваются способы получения трассы исполнения программы на основе инструментации кода программы: статическая инструментация исходного кода программы, статическая инструментация в процессе компиляции программы в исполняемый код, статическая инструментация исполняемого кода программы, динамическая инструментация кода программы,

инструментация интерпретирующей среды для интерпретируемых языков программирования, полносистемная эмуляция. Рассматриваются методы преобразования потока операций в программе в символьную формулу и этапы преобразования символьной формулы для подготовки вычисления новых наборов входных данных программы. Рассматриваются алгоритмы выбора пути для последующего анализа. Рассматриваются методы обнаружения ошибок в программе на основе регистрации аварийного завершения программы и на основе проверки нарушения предиката безопасности операции, исполнение которой потенциально может привести к проявлению ошибки. Сформулированы условия обнаружения ошибок и из последствия для следующих типов ошибок:

- *Деление на ноль (CWE-369)*;
- *Выход за границы буфера в памяти (CWE-119*;
- *Не верное завершение работы с ресурсом или освобождения ресурса (CWE-400, CWE-404*;
- *Использование памяти на куче после её освобождения (CWE-416)*;
- *Повторное освобождение памяти на куче (CWE-415)*;
- *Разыменование нулевого указателя (CWE-476)*;
- *Использование неинициализированной переменной (CWE-467)*.

В третьей главе описываются результаты исследования, изложенные в диссертации: модель обнаружения ошибок, алгоритм комбинированного анализа и метод классификации предупреждений об ошибках в программе.

Сначала описывается модель обнаружения ошибок в программе на основе методов символьного исполнения программы:

Определим программу, как четверку:

$$P = <F, S, s_I, S_T>, \quad (1)$$

где: S – множество состояний программы; s_I – начальное состояние программы; S_T – множество конечных состояний программы; F – множество операций, каждая из которых переводит программу из одного состояния в другое:

$$f: S \to S, f \in F, \quad (2)$$

Тогда исполнение программы можно определить как последовательность переходов между состояниями программы $s_i \in S$, осуществляемых операциями $f_i \in F$:

$$\{ f_1(s_1) \to s_1, f_1(s_1) \to s_2, \dots , f_i(s_i) \to s_T \}, \quad (3)$$

где: $s_1 \in S$ – начальное состояние программы, $s_T \in S_T$ – конечное состояние программы, а $f_i \in F$ – множеству операций программы, переводящих одно состояние программы в другое.

Определим состояние программы как,

$$s = <d, f> \mid d \in D,\ f \in F, \quad (4)$$

где: D – множество данных, обрабатываемых программой, f – следующая операция программы. Тогда исполнение операции программы можно представить в виде

$$f_i(d_i) \to < d_j, f_j > \mid \mathrm{d_i}, \mathrm{d_j} \in D,\ f_i, f_j \in F, \quad (5)$$

Определение 1. *Ограничением пути исполнения в программе PC* будем называть множество ограничений на значения данных программы, полученное путём преобразования операций, выполненных над данными программы на пути исполнения, в элементы множества ограничений и однозначно описывающее исполнение программы по пути, достигающем состояния s.

Не ограничивая общности рассуждений, всё множество операций в программе можно разделить на три вида (6):

$$f_i(d_i) \to < d_j, f' > \left| \begin{cases} d_i \neq d_j,\ f_i \to f_j & \text{вычислительная операция} \\ d_i = d_j,\ f_i \to f_j & \text{безусловный переход} \\ d_i = d_j, \begin{cases} f_i \to f_j \mid p_i \\ f_i \to f_k, \mid \neg p_i \end{cases} & p_i \in PC \text{ условный переход} \end{cases} \right. , \quad (6)$$

где:

- вычислительная операция изменяет состояние программы путём изменения множества данных программы $d_i \to d_j$ и переводит исполнение программына следующую операцию $f_i \to f_j$;
- безусловный переход изменяет состояние программы путём перевода исполнения на следующую операцию $f_i \to f_j$;
- условный переход изменяет состояние программы путём перевода исполнения на операцию f_j, если подмножество предусловий состояни, являющееся условием перехода, вычисляется в истину, и на операцию f_k, если подмножество

предусловий состояния, являющееся условием перехода, вычисляется в ложь.

Разделим множество данных программы D, на котором определено ограничение пути исполнения в программе PC, на два множества: V – множество внутренних данных программы; W – множество внешних данных программы. Тогда предусловие состояния s_i можно описать как функцию:

$$p(w, v) \mid w \subseteq W, v \subseteq V, \quad (7)$$

Заменим элементы множества W на элементы множества переменных X, где позиции каждого элемента множества внешних данных программы соответствует переменная $x \in X$.

Определение 2. *Множеством свободных переменных* будем называть множество переменных, значения которых соответствуют значениям элементов множества внешних данных программы.

Определение 3. *Множество зависимых переменных* формируется из переменных, являющихся результатом исполнения операций, множество аргументов которых содержит хотя бы одну свободную или хотя бы одну зависимую переменную.

Определение 4. *Символьное ограничение пути SPC* или символьное *предусловие состояния s* в программе, является множеством ограничений на значения внутренних данных программы, зависимых и свободных переменных и внутренних данных программы, полученных путём преобразования операций над ними на пути исполнения программы, предшествующем состоянию s.

Множество значений данных программы, удовлетворяющих символьному предусловию пути для состояния s_i, описывается функцией $\pi(x_i, v_i)$:

$$D_{\pi i} = \pi(x_i, v_i) \mid D_{\pi i} \subseteq D \quad (8)$$

Определение 5. *Состояние ошибки* это такое состояние программы $s_{err} \in S$, при котором дальнейшее исполнение программы ошибочно.

Определение 6. *Множество определения операции f* – это такое подмножество данных программы $D' \subseteq D$, на котором выполнение операции f не приводит к достижению состояния ошибки.

Утверждение 1. Для каждой операции f входящей в множество операции программы F существует множество определения данной операции.

$$\forall f \in F \exists D' \subseteq D, \quad (9)$$

Утверждение 2. Если множество значений аргументов операции f не входит в множество определения операции D' и множество D' не пусто, то исполнение операции приводит программу в состояние ошибки.

$$f(d) \rightarrow s_{err} \mid \forall\, d \notin D' \,\&\, D' \neq \varnothing, \quad (10)$$

Теорема 1. Если множество определения D' операции f_i не пусто и дополнение множества D' и множества значений данных программы $D_{\pi i}$, ограниченных функцией $\pi\,(x_i,\ v_i) = D_{\pi i}$ не пусто, то существуют такие значения значений внешних данных, исполнение программы на которых приведёт к достижению ошибочного состояния s_{err} в программе:

$$f_i(d_i) \rightarrow s_{err} \mid \forall\, d_i \in D_{err} = D_{\pi i} \setminus D' \,\&\, D' \neq \varnothing \quad (11)$$

Следствие теоремы 1. Для того чтобы вычислить внешние данные программы, приводящие исполнение программы в состояние ошибки, необходимо и достаточно для каждого типа ошибки определить множество операций, исполнение которых может приводить к проявлению ошибки, и сформулировать условие выхода значений аргументов операции за пределы множества определения этих операций и при этом входящих в множество значений данных программы.

После этого иллюстрируется применение *теоремы 1* на примерах обнаружения ошибок различного типа: формулируются и доказываются теоремы, опирающиеся на *теорему 1*, для обнаружения ошибок нескольких типов.

Ошибка деления на ноль

Определение *7. Операция целочисленного деления* – операция от двух аргументов, множество допустимых значений аргумента, соответствующего делимому, определено на всем множестве целых чисел, а множество допустимых значений аргумента, соответствующего делителю, определено на всём множестве целых чисел, за исключением элемента, соответствующего нулю:

$$f_{div}(d_{dividend}, d_{divisor}) \mid d_{dividend} \in D_{numbers},\ d_{divisor} \in D_{numbers} \setminus D_0, \quad (12)$$

где:

$d_{dividend}$ – множество значений делимого;

$d_{divisor}$ – множество значений делителя;

$D_{numbers}$ – множество целых чисел;

D_0 – множество чисел, содержащее один элемент, равный нулю.

Теорема 2. Для определения ошибки деления на ноль для операции целочисленного деления необходимо и достаточно добавить в символьное предусловие операции f_{div} ограничение множества значений переменной, соответствующей делителю $x_{divisor}$, условие равенства нулю элементов этого множества:

$$f_{div}(d_{dividend}, x_{divisor}) \rightarrow s_{err} \mid d_{dividend} \in D_{numbers}\ ,\ (SPC\ \&\ (x_{divisor} = 0)) \neq \varnothing \quad (13)$$

Ошибки работы с указателями на адрес в памяти

Определение 8. *Указателем* будем называть переменную в программе, значение которой соответствует адресу в памяти.

Определение 9. Операцией *разыменования указателя* будем называть операцию f_{deref} взятия значения данных программы по адресу в памяти, заданного как аргумент операции разыменования. Областью определения операции разыменования D_{deref} являются адреса памяти, доступные программе.

Теорема 3. Для определения ошибки разыменования нулевого указателя необходимо и достаточно в символьное предусловие операции разыменования f_{deref} добавить условие равенства значения переменной указателя нулю:

$$f_{deref}(x) \rightarrow s_{err} \mid (SPC\ \&\ (x = 0)) \neq \varnothing, \quad (14)$$

Определение 10. *Буфер в памяти* – область памяти, ограниченная адресом начала A_{begin} и адресом конца A_{end} блока данных программы в памяти.

Определение 11. Операцией *разыменования при доступе к буферу* в памяти будем называть операцию разыменования указателя, область определения которой D'_{buffer} ограничена адресом начала A_{begin} и адресом конца A_{end} данных программы в памяти.

Теорема 4. Для определения ошибки доступа к буферу в памяти по указателю f_{dbuf} необходимо и достаточно в символьное предусловие операции добавить условие проверки значения переменной указателя меньше адреса начала буфера A_{begin} и больше адреса конца буфера A_{end}:

$$f_{dbuf}(d) \rightarrow s_{err} \mid (SPC\ \&\ (d < A_{begin}\ \ ||\ d > A_{end})) \neq \varnothing, \quad (15)$$

Также формулируются и доказываются теоремы определения ошибок доступа к освобожденному блоку памяти, использования

неинициализированных переменных и неправильной работы с ресурсом операционной системы.

Далее в главе приводится *алгоритм 1* комбинирования статического и динамического анализа с целью вычисления входных данных, приводящих к реализации ошибки в программе, который состоит из следующих шагов:

1. Обнаружение потенциальных ошибок в программе методами статического анализа.
2. Статический анализ машинного кода программы для получения сокращённого графа потока программы с целью вычисления путей, содержащих трассу событий для каждой найденной статическим анализом ошибки.
3. Вычисление расстояния от каждого базового блока программы, лежащих на путях, вошедших в сокращённый граф потока программы, до точки на трассе событий для каждой ошибки, найденной с помощью статического анализа.
4. Реализация направленного динамического символьного исполнения программы с целью генерации входных данных для проведения исполнения программы по путям, достигающим трассу ошибки.
5. Генерация внешних данных программы, нарушающих предикат безопасности операции в точке проявления ошибки.
6. Проверка проявления ошибки в процессе исполнения программы на внешних данных программы, сгенерированных на этапе 5.

Для реализации данного алгоритма приводится решение задачи построения сокращённого графа потока, в котором исключены пути, не содержащие трассу предупреждения об ошибке. Для ошибок, найденных при помощи статического анализа исходного кода программы, производится сопоставление трассы ошибки в исходном коде с трассой ошибки в исполняемом коде. Показывается возможность наличия большего количества путей исполнения в исполняемом коде, связанное с необходимостью вычисления сложных условных выражений. Далее рассматривается *алгоритм 2* построения сокращенного графа потока программы, содержащего пути, достигающие трассу событий предупреждения об ошибке:

1. Сначала строится *сокращённый граф вызовов подпрограммы*. Построение начинается с подпрограмм, в которых находятся

точки проявления ошибки в программе. Далее граф дополняется подпрограммами, вызывающими подпрограммы, содержащие точки проявления ошибки в программе. Построение графа продолжается до тех пор, пока не будет достигнута подпрограмма, содержащая точку входа в программу или главную функцию вычислительного потока. Таким образом из машинного кода программы извлекается *сокращённый граф вызовов*, содержащий только те подпрограммы, исполнение которых необходимо для проявления ошибок, предупреждения о которых получены от инструмента статического анализа программ.

2. Далее для подпрограмм, вошедших в *сокращенный граф вызовов*, производится статическое извлечение *сокращённого межпроцедурного графа потока*, содержащего пути исполнения, достигающие трассу предупреждения об ошибке, полученного от инструмента статического анализа программ.
3. После получения *сокращённого межпроцедурного графа потока программы* производится вычисление числовой метрики расстояния, выражающейся в количестве условных переходов от каждого базового блока программы до каждой точки трассы событий предупреждения об ошибке в программе. В дальнейшем расстояние от базового блока до точки в программе используется как численная оценка при выборе следующего пути исполнения для анализа: сначала выбираются пути, для которых эта оценка меньше.

Далее описывается *алгоритм 3* направленного символьного исполнения, который, используя эвристику минимального расстояния, производит динамическое символьное исполнение программы с целью вычисления входных данных, проводящих исполнение программы по трассе ошибки:

1. Производится топологическая сортировка точек событий на трассе предупреждения об ошибке на графе потока программы.
2. На первой итерации анализа программа запускается на выполнение с произвольным набором внешних данных.
3. Производится сбор трассы исполнения программы и преобразование её в символьную формулу, в которой

внешние данные программы представлены свободными (символьными) переменными.

4. Если путь исполнения в программе не прошёл по трассе событий предупреждения об ошибке в программе, то производится инвертирование условия для условного перехода, находящегося в базовом блоке, с наименьшей оценкой расстояния до следующей точки на трассе событий предупреждения об ошибке в программе.
5. Вычисляется новый набор внешних данных программы с целью проведения исполнения программы по новому пути.
6. Производится запуск программы на новом наборе внешних данных и выполнение алгоритма продолжается с этапа 3 до тех пор, пока следующая точка на трассе событий предупреждения об ошибке не будет достигнута или не останется путей, достигающих следующую точку на трассе предупреждения об ошибке.
7. Работа алгоритма завершается при достижении последней точки трассы событий предупреждения об ошибке – точки проявления ошибки.

Далее в главе говорится о необходимости добавления к условию пути инвертированного предиката безопасности для проверки возможности вычисления внешних данных программы с целью проявления ошибки. Приводится классификация предупреждений об ошибках на три класса: подтвержденных предупреждений, для которых вычислены входные данные и на них проявляется ошибка; неподтвержденные предупреждения, для которых доказана несовместность условия на трассе предупреждения; предупреждения, для которых не удалось подобрать внешние данные программы, проводящие её исполнение по трассе предупреждения об ошибки и не удалось доказать несовместность условий на трассе предупреждения и Заканчивается глава

В четвертой главе описывается реализация программной системы и эксперимент, подтверждающий применимость предложенного метода анализа программ. Программная система построена на основе инструментов *Svace* и *Anxiety*, разработанных в Институте системного программирования им. В.П. Иванникова Российской академии наук. Описывается модульная структура

инструмента *Anxiety* и назначение каждого модуля для реализации направленного динамического символьного исполнения, которая позволяет:

- производить замену реализации каждого модуля без изменения общего алгоритма динамического символьного исполнения
- даёт возможность проведения анализа в параллельном и распределенном режиме.

Описывается реализация модулей, а также ограничения, накладываемые реализацией на класс анализируемых программ. Указываются ограничения реализации алгоритмов, в частности отсутствие реализации алгоритма проверки совместности условий трассы предупреждения об ошибке, в следствие чего невозможность классификации предупреждений об ошибке как ложноположительных вследствие ограничений реализации.

Описываются методы оценки инструментов анализа программ, основанные на:

- использовании приемочных тестов для инструмента;
- использовании сертифицированных наборов оценки инструментов анализа программ;
- результатах анализа проектов с открытым исходным кодом.

Проверка предложенного метода производилась на наборе из 12 программных утилит из поставки ОС Debian Linux. Исходный код программ анализировался инструментом *Svace*, после чего проводилось вычисление входных данных для найденных ошибок при помощи инструмента Anxiety. Результаты вычисления входных данных приведены в таблице 1:

Вид ошибки	Построены данные
Переполнение буфера	7
Повисший указатель	25
Разыменование после проверки на NULL	13
Разыменование NULL	6
Утечка дескриптора ресурса	2
Утечка памяти	9
Переполнение строки	2
Использование памяти после освобождения	1
Использование дескриптора после закрытия	1

Таблица 1 Результаты проверки предложенного метода

В процессе построения входных для прохождения по трассе предупреждений об ошибках в программе инструмент построил 26597 уникальных входных данных, которые привели к аварийному завершению программы по 24 уникальным адресам, из которых 2 адреса соответствовали вызову функции принудительного останова работы программы *abort()* (SIGABRT) и 22 ошибке сегментации (SIGSEGV), что соответствует ошибки работы с памятью. Для 13 из 24 адресов удалось восстановить строку исходного текста программы ни для одной из них не было обнаружено предупреждений об ошибке от инструмента статического анализа. Наличие данных ошибок в программе является одной из причин невозможности достижения трассы предупреждений об ошибках в связи с аварийным завершением программы до достижения точки проявления ошибки.

Проводится анализ результатов экспериментальной проверки и делается вывод, что несмотря на известные ограничения инструмента динамического символьного исполнения удалось построить внешние данные программы для прохождения по трассе событий предупреждения об ошибке для нескольких типов ошибок, что подтверждает возможность классификации предупреждений об ошибках предложенным методом.

В заключении формулируются основные результаты диссертационной работы и приводятся направления дальнейших исследований, направленных на увеличение точности отслеживания помеченных данных программы путём поддержки неявных зависимостей по данным и по управлению, выполнить реализацию проверки условий для разных классов ошибок и поддержку анализа многопоточных и интерактивных программ.

Основные результаты диссертационной работы:

- разработана модель обнаружения ошибок в программе в процессе символьного исполнения программы;
- разработаны алгоритмы комбинированного анализа программ на основе направленного динамического символьного исполнения программ, использующего предупреждение об ошибке, полученное методами

статического анализа исходного кода программ, для вычисления внешних данных программы, приводящих к проявлению программной ошибки в процессе исполнения программы;

- разработан метод классификации предупреждений об ошибках в программе, основанный на применении разработанных модели и алгоритмов комбинированного анализа программ.

Список публикаций автора по теме диссертации

1. Герасимов А. Ю., Круглов Л. В. Вычисление входных данных для достижения определённой функции в программе методом итеративного динамического анализа / А. Ю. Герасимов, Л. В. Круглов // Труды ИСП РАН. 2016. Т. 28, вып. 5. С. 159-174.
2. Герасимов А. Ю. Обзор подходов к улучшению качества результатов статического анализа программ // Труды ИСП РАН. 2017. Т. 29, вып. 3. С. 75-98.
3. Герасимов А. Ю. Подход к определению достижимости программных дефектов, обнаруженных методом статического анализа, при помощи динамического символьного исполнения / А. Ю. Герасимов, Л. В. Круглов, М. К. Ермаков, С. П. Вартанов // Труды ИСП РАН. 2017. Т.29, вып. 5. С. 111-134.
4. Gerasimov A. Reachability confirmation of statically detected defects using dynamic analysis / A. Gerasimov, L. Kruglov // Proceedings of the 11th International Conference on Computer Science and Information Technologies (CIST). Yerevan, 2017.
5. Герасимов А. Ю. Об ограничениях классификации дефектов в программах, найденных методами статического анализа программ при помощи динамического символьного исполнения // 60-я конференция МФТИ. Долгопрудный, 2017. С. 101-103.
6. Gerasimov A. Anxiety: a dynamic symbolic execution framework / A. Gerasimov, S. Vartanov, M. Ermakov, L. Kruglov, D. Kutz, A. Novikov, S. Astyan // Ivannikov ISPRAS Open Conference - 2017. Moscow, 2017.
7. Gerasimov A. Reachability confirmation of statically detected defects using dynamic analysis / A. Gerasimov, L. Kruglov //

Proceedings of the Ivannikov Memorial Workshop. Yerevan, Armenia, 3-4 May, 2018.
8. Герасимов А. Ю. Направленное динамическое символьное исполнение программ для подтверждения ошибок в программах // Программирование. 2018. №5. С. 316-323.
9. Gerasimov A. Y. An approach to Reachability Determination for Static Analysis Defects with the Help of Dynamic Symbolic Execution / A. Y. Gerasimov, L. V. Kruglov, M. K. Ermakov, S. P. Vartanov // Programming and Computer Software. 2018. Vol. 44. No. 6. P. 443-451.
10. Свидетельство о государственной регистрации программы для ЭВМ 2016660242. Российская Федерация. Инструмент итеративного динамического анализа программ / А. Ю. Герасимов. - заявл. 12.07.2016; зарегистрир. 08.09.2016; опубл. 20.10.2016.
11. Свидетельство о государственной регистрации программы для ЭВМ 2016660244. Российская Федерация. Инструмент вычисления наборов входных данных для достижения определенной инструкции в программе / А. Ю. Герасимов. - заявл. (12.07.2016); зарегистрир. 09.09.2016; опубл. (20.10.2016).
12. Свидетельство о государственной регистрации программы для ЭВМ 2017660034. Российская Федерация. Anxiety: модуль направленного анализа для инструмента итеративного динамического символьного исполнения программ / А. Ю. Герасимов. - заявл. (17.07.2017); зарегистрир. 13.09.2017; опубл. (13.09.2017).
13. Свидетельство о государственной регистрации программы для ЭВМ 2017660037. Российская Федерация. Anxiety: модульный инструмент итеративного динамического символьного исполнения программ / А. Ю. Герасимов. - заявл. (17.07.2017); зарегистрир. 13.09.2017; опубл. (13.09.2017).
14. Свидетельство о государственной регистрации программы для ЭВМ 2017660154. Российская Федерация. Anxiety: модуль параллельных вычисления для инструмента динамического символьного исполнения / А. Ю. Герасимов. - заявл. (17.07.2017); зарегистрир. 18.09.2017; опубл. (18.09.2017).