"УТВЕРЖДАЮ"

льного Исследовательского

нформатика и Управление»

кадемик РАН И.А.Соколов

Отзыв ведущей организации на диссертацию

Ермакова Михаила Кирилловича

Методы повышения эффективности итеративного динамического анализа программ

представленную на соискание ученой степени

кандидата технических наук по специальности

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

Актуальность

В настоящее время разработка программного обеспечения является активно развивающейся и востребованной областью. Сложность создаваемых программных комплексов повышается, а требования к их надёжности возрастают, что в значительной степени затрудняет сам процесс разработки. Для решения подобных проблем используются инструменты, позволяющие полностью или частично автоматизировать этапы процесса разработки. Среди подобных инструментов можно выделить группу программных средств, осуществляющих проверку качества программного кода. Традиционно выделяют следующие две обширные группы подходов к исследованию качества программ: статический анализ и динамический анализ. При проведении статического анализа не производятся запуски исследуемой программы на выполнение — инструменты статического анализа автоматически создают структуры данных, описывающие код программы в различных представлениях (например, абстрактное синтаксическое дерево или граф потока управления) и осуществляют обработку этих структур. Динамический анализ заключается в исследовании программ во время их выполнения. Это позволяет исследовать программы, если инструменту анализа доступен исполняемый код программы, но не доступен её исходный код. Методы

статического анализа хорошо масштабируемы, однако зачастую имеют высокий уровень ложных срабатываний. Методы динамического анализа позволяют исследовать поведение программы при её выполнении на конкретных наборах входных данных, что практически полностью устраняет ложные срабатывания. В то же время, динамический анализ позволяет проверять только те фрагменты кода программы, которые были реально выполнены при проведении анализа. Получение точной оценки качества кода программы с помощью методов динамического анализа обычно требует множественных запусков программы на выполнение на различных наборах входных данных. Эта особенность динамического анализа делает крайне актуальными методы, позволяющие автоматически строить наборы входных данных для исследования различных путей выполнения программы. Для реализации подобного подхода могут быть использованы принципы символьного исполнения программ. Символьное исполнение позволяет связывать пути выполнения программы с входными данными с помощью наборов булевых формул. Запуск программы на наборе входных данных позволяет исследовать путь выполнения и получить для него трассу ограничений, состоящую из указанных выше формул. Ключевым ограничением систем, осуществляющих автоматический обход путей выполнения программ на основе символьного исполнения, является чрезвычайно высокая вычислительная сложность этого подхода. Количество путей выполнения в программах растёт экспоненциально с увеличением объёма кода, а задача проверки выполнимости булевых формул не имеет в настоящее время полиномиальных алгоритмов решения. Подобные ограничения делают крайне актуальными исследования, направленные на разработку механизмов повышения эффективности динамического анализа, включающего автоматический обход путей выполнения программы на основе символьного исполнения. Добиться подобного ускорения можно путём обхода только части путей выполнения программы. При наличии знаний о структуре анализируемой программы возможно сформулировать критерии выбора путей выполнения программы таким образом, чтобы проводить целенаправленное исследование отдельных фрагментов кода. Существующие инструменты обхода путей выполнения программы на основе символьного исполнения либо не предоставляют подобные возможности частичного анализа, либо проведение частичного анализа с их помощью накладывает дополнительные ограничения на анализируемые программы. В то же время, проведение частичного анализа представляется актуальным в процессе разработки программ — настройку анализа может производить разработчик или тестировщик, обладающий знаниями о внутреннем устройстве программы и

особенностях входных данных. Среди инструментов автоматического обхода путей выполнения программы на основе символьного исполнения для анализа непосредственного выполнения программы используются системы инструментации исполняемого кода. Инструментация кода программы подразумевает его модификацию путём внедрения кода, который не нарушает исходную функциональность программы и позволяет извлекать дополнительную информацию при выполнении программы. Предварительная инструментация исполняемого кода не требует наличия исходного кода программы и позволяет снизить накладные расходы по сравнению с динамической инструментацией исполняемого кода. Во-первых, это достигается за счёт того, что разбор кода программы и его модификация производятся однократно, в то время как один и тот же фрагмент кода может выполняться на большом количестве путей. Во-вторых, предварительная инструментация проводится независимо от выполнения программы, что позволяет более эффективно распределять рабочую нагрузку при использовании нескольких вычислительных узлов. Возможность распределения нагрузки на несколько вычислительных узлов при использовании предварительной инструментации является крайне актуальной в настоящее время при анализе программ, запускаемых на мобильных устройствах (например, платформ Android/ARM и Tizen/ARM), обладающих меньшим объемом ресурсов по сравнению с традиционными аналогами.

В качестве базового инструмента для проведения исследований в рамках диссертационной работы был выбран инструмент Avalanche, разрабатываемый в ИСП РАН.

Целью диссертационной работы является разработка и реализация методов увеличения эффективности динамического анализа программ, включающего автоматический обход путей выполнения программ на основе символьного исполнения, с помощью ограничения количества исследуемых путей выполнения по пользовательским спецификациям и применения статической инструментации исполняемого кода для извлечения трасс выполнения программ. В работе решены **следующие задачи**:

1. Проведен обзор методов динамического анализа программ на основе символьного исполнения и методов статической инструментации кода.
2. Выбраны инструментальные средства, на базе которых проводится разработка и реализация предлагаемых в работе методов.

3. Разработан и реализован метод частичного анализа программ, включающий автоматический обход подмножества путей выполнения программы на основе символьного исполнения, где подмножество путей задаётся с помощью пользовательских спецификаций.
4. Разработана и реализована схема параллельной обработки независимых путей выполнения в рамках выбранного средства динамического анализа с целью повышения эффективности использования вычислительных ресурсов.
5. Разработан метод статической инструментации исполняемого кода.
6. На основе этого метода реализована программная система для платформы ARM/Linux. Система содержит модули построения трасс выполнения программы для использования в рамках выбранного средства динамического анализа.
7. Проведена оценка эффективности разработанных методов.

Исходная реализация инструмента Avalanche, используемого в качестве базового в рамках работы, не учитывает особенности доступных вычислительных ресурсов, что делает актуальной задачу расширения функциональности инструмента.

Основные результаты:

- Реализована система статической инструментации исполняемого кода, которая поддерживает:
  - возможности инструментации на уровне отдельных инструкций с извлечением параметров данных инструкций. Эта функциональность необходима для внедрения в программу дополнительных инструкций с целью проведения символьного исполнения;
  - пользовательские спецификации точек инструментации и инструментационного кода. Эта функциональность позволит предоставить возможности создания инструментов анализа, сравнимые с рассмотренными существующими средствами, активно применяемыми на практике.
- Реализованы следующие модификации инструмента Avalanche:
  - Определяется язык спецификаций, позволяющий задавать статические множества интервалов смещений.
  - Пользователь получает возможность задавать спецификации для каждого источника внешних данных, обрабатываемого целевой программой.

- Добавлены дополнительные проверки в алгоритм работы инструмента tracegrind.
- Обновлено отображение для ячеек памяти и регистров программы и запись ограничений на символьные переменные.
- Определяется язык спецификаций, позволяющий указывать множества имён функций, точки ветвления в которых необходимо обрабатывать, и множество имён функций, точки ветвления, в которых необходимо пропускать. Пользователь задаёт эти спецификации при запуске инструмента.

- Разработана схема проведения параллельных вычислений в рамках отдельных итераций, основываясь на независимости обработки подтрасс и соответствующих им наборов входных данных. Предлагаемая схема параллельной обработки путей выполнения учитывает особенности потоков данных и подзадач анализа, проводимого инструментом Avalanche. В рамках этой схемы рассматривается модификация управляющего модуля инструмента Avalanche, позволяющая проводить работу с параллельными вычислителями, выполняющими задания из банка заданий, строящегося на каждой итерации анализа. Практическая реализация схемы включает созданный в рамках работы управляющий модуль инструмента Avalanche расширенной функциональности. Применение разработанной схемы на практике для анализа ряда проектов с открытым исходным кодом позволило увеличить эффективность анализа на вычислительном устройстве многоядерной архитектуры по сравнению со стандартной версией инструмента Avalanche. Было зафиксировано ускорение обхода путей выполнения и обнаружены дополнительные дефекты, приводящие к аварийному завершению программ.

Научная новизна

В рамках диссертационной работы получены следующие результаты, обладающие научной новизной:

1. Предложен метод частичного динамического анализа программ, включающего автоматический обход подмножества путей выполнения программы, задаваемого пользовательскими спецификациями на источники входных данных и точки ветвления в коде программы. Предложенный метод не накладывает ограничений на формат входных данных программы и не требует непосредственного доступа к исходному коду программ.

2. Предложен метод статической инструментации исполняемого кода для извлечения трасс выполнения программы с целью построения наборов входных данных и оценки их приоритетности в рамках метода автоматического обхода путей выполнения программы.

Практическая значимость

Предложены методы, повышающие эффективность динамического анализа программ, включающего автоматический обход путей выполнения программ на основе символьного исполнения, и методы статической инструментации исполняемого кода. Предложенные методы и полученные в процессе разработки методов наблюдения могут быть включены в курсы анализа программ, автоматизации тестирования и компиляторных технологий, преподаваемых в высших учебных заведениях, специализирующихся на исследовании информационных технологий. Реализация предложенных методов проведена в рамках инструментальных средств с открытым исходным кодом, что позволяет использовать эти методы при решении задач анализа программ независимым исследователям и при решении прикладных и промышленных задач анализа конкретных программных продуктов разработчикам и тестировщикам.

Замечания.

- Более половины текста занимают разного рода обзоры.
- Во многих местах текст подготовлен небрежно (например, рисунок 16, стр. 53, строки 5-6).
- Текст не вычитан, много опечаток и замечаний по русскому языку.
- По результатам анализа систем инструментации программ делается вывод о том, что наиболее подходящим средством является система Avalanche, разработанная в ИСП РАН. Для обоснования этого выбора приводится таблица сравнительных характеристик разных систем (таблица 1), из которой выбор системы Avalanche совершенно не следует. Понятно, что система Avalanche более подходит для использования в данном контексте, поскольку она разработана в ИСП РАН, и это было бы серьезным основанием.
- Отсутствует формальное обоснование принятых решений, все делается на примерах, часто излишне детальных, вплоть до байтов.
- Одним из направлений разработки является возможность взаимодействия с пользователем. Упоминается, например, язык спецификаций, позволяющий указывать множества имён функций. В работе вообще про интерфейс с пользователем ничего не говорится.
- В таблице 4 (стр. 75) приведены результаты применения инструмента Avalanche для обработки целевых программ при использовании стандартной схемы и при

использовании нескольких (4) потоков выполнения. При использовании 4 параллельных потоков выполнения удалось добиться увеличения количества пройденных путей выполнения и обнаружения дополнительного количества критических дефектов. Непонятно, почему именно 4? А что будет в случае другого числа потоков?

Перечисленные замечания не влияют на положительную оценку диссертационной работы и не ставят под сомнение полученные в ней результаты. Результаты диссертации своевременно опубликованы. Автореферат полно и правильно отражает содержание диссертации.

Диссертация Ермакова М.К. является законченной научно-исследовательской работой и удовлетворяет всем требованиям ВАК, предъявляемым к диссертациям на соискание ученой степени кандидата технических наук по специальности 05.13.11 – математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей, а Ермаков Михаил Кириллович заслуживает присуждения ученой степени кандидата технических наук по специальности 05.13.11.

Отзыв на диссертацию составлен доктором физико-математических наук Серебряковым Владимиром Алексеевичем и обсужден на семинаре отдела систем математического обеспечения Вычислительного центра им. А.А.Дородницына ФИЦ ИУ Российской академии наук 11.11.2016 протокол №1.11.16.

Зав. отделом ВЦ РАН

д.ф.м.н., профессор В. А. Серебряков